AD-784 379

A GUIDE FOR THE APPLICATION OF PERFORMANCE-STRUCTURE ORIENTED CAI IN NAVAL TRAINING: A WORKING PAPER

Joseph W. Rigney, et al

University of Southern California

Prepared for:

Naval Training Equipment Center
Advanced Research Projects Agency

July 1974

Technical Report: NAVTRAEQUIPCEN 73-C-0065-1

A GUIDE FOR THE APPLICATION OF PERFORMANCE-STRUCTURE ORIENTED CAI IN NAVAL TRAINING

ABSTRACT

Considerations and procedures for applying Performance-Structure Oriented CAI in Naval training are described, in terms of a general diagram of the necessary elements in a CAI system.

The fundamental objective is to lead the student to develop his own cognitive structures that will serve him for generating the surface structures of tasks. The instructional system must accomplish this by generating and scheduling a suitable instructional sequence for each student.

The developmental steps, from job task-structure analyses to computer programs, are described. Techniques for generalizing these procedures fall into two categories: higher-level programming languages that would permit more rapid production of programs for specific applications, and general-purpose programs that accept data modules specific to an application. Both kinds of techniques have been used in earlier applications.

Examples of instructional flowcharts, programmer's flowcharts, and data encoding procedures used in the development of the RIO trainer are presented to illustrate key points in the developmental steps.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Naval Training Equipment Center under Contract N61339-73-C-0065.

UNCLASSIFIED
Security Classification

AD-784379

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) University of Southern California University Park Los Angeles, California 90007	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE
A GUIDE FOR THE APPLICATION OF PERFORMANCE-STRUCTURE ORIENTED CAI IN NAVAL TRAINING: A WORKING PAPER

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report

5. AUTHOR(S) (First name, middle initial, last name)
Joseph W. Rigney, D. Kirk Morrison, and Louis A. Williams

6. REPORT DATE July 1974	7a. TOTAL NO. OF PAGES 66	7b. NO. OF REFS 53
8a. CONTRACT OR GRANT NO. N61339-73-C0065-1	9a. ORIGINATOR'S REPORT NUMBER(S) TR 73	
b. PROJECT NO. 2752-02P02		
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) NAVTRAEQUIPCEN 73-C-0065-1	
d.		

10. DISTRIBUTION STATEMENT
Approved for public release; distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Training Equipment Center Orlando, Florida 32813

13. ABSTRACT

Considerations and procedures for applying Performance-Structure Oriented CAI in Naval training are described, in terms of a general diagram of the necessary elements in a CAI system.

The fundamental objective is to lead the student to develop his own cognitive structures that will serve him for generating the surface structures of tasks. The instructional system must accomplish this by generating and scheduling a suitable sequence for each student.

The developmental steps, from job task-structure analyses to computer programs, are described. Techniques for generalizing these procedures fall into two categories: higher-level programming languages that would permit more rapid production of programs for specific applications, and general-purpose programs that accept data modules specific to an application. Both kinds of techniques have been used in earlier applications.

Examples of instructional flowcharts, programmer's flowcharts, and data encoding procedures used in the development of the RIO trainer are presented to illustrate key points in the developmental steps.

This research is sponsored by Advanced Research Projects Agency, ARPA Order No. 2310, Program Code No. 3D20.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6801

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Assisted Instruction						
Performance-Structure Oriented CAI						
Procedures for Developing CAI Applications						

DD FORM 1 NOV 65 1473 (BACK)
(PAGE 2)

UNCLASSIFIED
Security Classification

FOREWORD

The University of Southern California and the Naval Training Equipment Center, under ARPA funding, are cooperating in an effort to develop a system which will serve to facilitate the production of appropriate Computer-Aided Instruction (CAI) for the Navy. This system, basically taking the form of a model or theory of CAI, is in part derived from and evaluated by empirical studies as described in other Naval Training Equipment Center technical reports by the present authors (Rigney, et. al., 1973; Rigney, et. al., in preparation). Because this empirical aspect of the research exhausts major portions of the resources available for the project, explication of the system contained herein is in its earliest stages. It is published here mainly for heuristic reasons and is not intended to be used as a refined set of guidelines for developing CAI materials.

Even in this initial form, however, the system can suggest to course authors various components of CAI and gross steps associated with the development of these components that need to be considered in the process of course construction. Further development of the system will consist of endeavors to expand upon current capabilities by offering: (a) specific instructional approaches for the components forming the structure of CAI; (b) computer programs, capable of implementing the suggested instructional approaches, which are general enough to apply in a range of subject matter areas; and (c) computer capabilities for generating some portions of the CAI specifications for a given application. Thus, CAI program developers will be able to use the system as an aid to deciding upon instructional approaches appropriate for their particular teaching objectives. Further, they even will be able to obtain computer programs which essentially are ready for application in their training program.

The development of generalizable instructional approaches and the supporting computer programs is a prime reason for viewing the extensive CAI course construction activities as essential to the project. The empirical research, however, serves additional important functions, among which is the ancillary contribution of developing cost-effective means for teaching skills for critical Navy jobs. The Radar Intercept Officer's job was the first technical area addressed in this way. Developing CAI materials for teaching the utilization of the AWG-9 system for maintaining the F-14 aircraft is being considered for the continuation of the empirical aspect of the research on this project.

Arthur S Blaiwes

ARTHUR S. BLAIWES
Scientific Officer

TABLE OF CONTENTS

Section		Page
I.	INTRODUCTION	1
II.	GENERAL CAI SYSTEM CONSIDERATIONS	4
	Internal Processing	4
	Student-Program Interface	4
	Student Data	6
	Adaptive Controller	6
	Instructional Sequence	8
	Feedback Loops	8
III.	PROCEDURES FOR GENERATING SPECIFICATIONS FOR A PSO CAI SYSTEM	9
	Categorization of Surface Structure Features	11
	Categorization of Content and L & M Mediators	12
	Implications of Task Analyses for CAI System Specifications	15
	Examples of Procedures for Generating Software Specifications	20
IV.	SUMMARY AND CONSLUSIONS	48
	REFERENCES	50
	APPENDIX A	54

LIST OF ILLUSTRATIONS

Figure		Page
1.	General Flowchart for PSC CAI Instructional Sequence . .	2
2.	Outline of Major Elements in CAI with Adaptive Controller	5
3.	Overall Performance Cycle	10
4.	Performance as a Goal-Action Hierarchy	10
5.	An example of the Propositional Representation of the Sentence	12
6.	Tentative Outline of Procedure for Generating CAI System Specifications from an Analysis of the Performance to be Taught	17
7.	Block Diagram of Hands-On Training CAI Testbed Hardware	18
8.	Preliminary Flowchart for Driving Interaction Cycle: Attack Phase, RIO Trainer	24
9.	Radar Intercept Officer Basic Instructional Trainer - Instructional Strategy Flow Diagram	25
10.	B-Scan Flow Diagram	28
11.	Example of a Student Sufficient History Used for Trials-to-Criterion Logic	30
12.	The RIO Trainer IS Scheduler Programmer's Flowchart . .	32
13.	Programmer's Flowchart for the Command Interpreter . . .	35
14.	Programmer's Flowchart for the COMP Subroutine	44
15.	The Problem Pointer Word Used in Generating Practice Problems	46

SECTION I

INTRODUCTION

A guide for generating specifications to apply Performance-Structure-Oriented (PSO) CAI (Rigney, et al., 1972a) in a variety of subject-matter areas in Naval technical training is the subject of this report. The planning activities required for developing a CAI application always are difficult and time-consuming. It would be useful to have guidelines to follow that would reduce the amount of work and that would assure more predictable outcomes. The guide to be described here is addressed to those objectives.

This is the second report of this series. In the first report (Rigney, 1973a) we described what we mean by the term, Performance-Structure-Oriented CAI, and listed some general steps in the development of PSO CAI. A bit of recapitulation is in order. PSO CAI was designed to give personnel, in technical schools or on the job, drill and practice in performing important tasks in their jobs. We have observed over the years that it is extremely difficult to assemble the resources in the right configuration to give personnel intensive practice in performing, under properly controlled conditions. For example, it is expensive to use operational equipment in electronics training schools, and this equipment lacks features that would immensely enhance its usefulness for training vis-a-vis operations. Intensive practice is a way of consolidating material that has been learned in the classroom, of forcing the student to organize this material in relation to performance requirements, and of insuring longer retention.

The approach used in PSO CAI is quite simple. We let the student "try out his wings" on a series of problems typical of the job he will be expected to do. The instructional environment includes instructional operators, which are features designed to assist the student in learning some particularly difficult skill, or in understanding some particularly difficult concept, or which give him advice about his current progress, or about what to do next. If the student cannot perform very well, he is switched to drills on subtasks or subskills in which he needs more practice. He then returns to the "top level" and attempts to perform at that level again. The data-processing technology allows us to record very detailed information about each student and to use that information to modify the instructional sequence to suit his individual needs. When the student can perform satisfactorily at the top level, which is representative of job requirements, according to criteria which the CAI system monitors, he has finished the course. The concept is illustrated in Figure 1. Under some circumstances, the entire student population may be deficient in subskills that are critical for job performance, and it may be more efficient to teach these subskills first, before entering the performance training section of the instructional system, in what some instructional technologists call a front-loading section. The diagram in Figure 1 indicates that more than one subskill or set of subskills may be so taught.

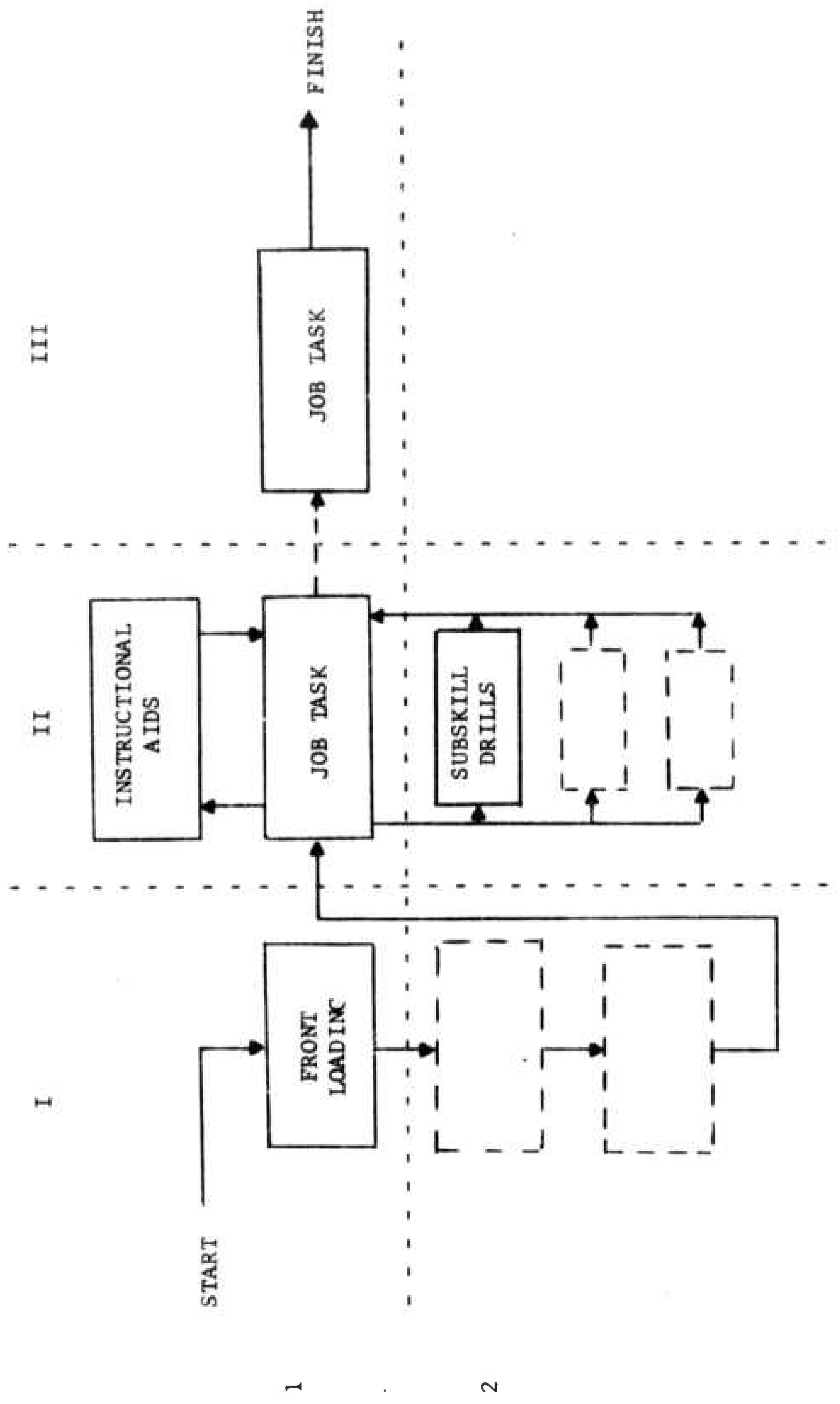


Figure 1. General Flowchart for PSO CAI Instructional Sequence

Variations in entering skills and knowledge are currently handled by trials-to-criterion logic. If a student can perform the easiest of a graded series of problems, he is transitioned to the next level of problems. Under certain circumstances, more powerful adaptive control techniques may be applicable; e.g., some variation of dynamic programming, a mathematical technique for optimization. Variations in learning rates and in patterns of essential subskills are currently handled by vertical iteration through hierarchically structured material, combined with trials-to-criterion logic. Within a block of problems, a student may, for example, be required to repeat a problem under conditions which unburden him of performing all but one of the subtasks required in the problem, so that he can concentrate on doing that one correctly.

This concept of how to train personnel to perform has been implemented in an individual skills trainer for the Radar Intercept Observer (Rigney, et al., 1973b) and in a program for giving students practice in troubleshooting electronic devices (Rigney, et al., 1972a).

The problem for the developmental model to be described here is to provide ways to specify the essential elements for applying PSO CAI in a particular context. To do this, it is necessary to know in great detail the characteristics of the performance that is to be the subject of the training. What is it that the student must learn to do?

When this has been described, the information can be used for generating specifications for a CAI system. In this guide, the diagram of the essential elements in a CAI system given in Figure 2 will be used to identify the "bases that must be touched" in the generation of these specifications. This diagram has been the subject of recent work on a "CAI testbed" concept (Rigney, 1973c).

SECTION II

GENERAL CAI SYSTEM CONSIDERATIONS

In this section, we will develop the considerations for determining the type of CAI system, in terms of particular examples of the elements, as outlined in Figure 2, that would be needed for applying PSO CAI in a selected context. In most cases, the different configurations could be produced by software, although there also could be requirements for special types of peripheral hardware in addition to standard terminals, to be interfaced with the basic processor. We will attempt here to lay out the considerations for each element in the system, so that procedures for generating specifications can be formulated in the next sections.

Internal Processing

This box represents the student. Of course, just as we do for other elements in the CAI system, we must develop "specifications" for the student. Implicit assumptions about student characteristics must be made explicit. There always is some definable population of students--some "target" population, for whom the CAI system is to be designed. We need to know characteristics of this population in relation to prerequisites for the course, and in terms of learning abilities. If we are going to adapt to individual differences among students with respect to entering skills and knowledge and with respect to rates of learning, we need to know something about these student characteristics.

As a form of drill and practice, PSO CAI assumes the students already are familiar with the "theory of operation" or are learning it in a parallel course. This form of CAI normally would be embedded in the total curriculum. Therefore, the screening that is done in the broader context should suffice for prerequisite requirements. Rate of learning might be predicted from a mixture of achievement and intelligence tests. However, it is preferable to use adaptive control mechanisms to sense important individual differences and to adjust to them as the student is progressing, rather than to depend upon the classical psychometric approach, viz, batteries of tests and multiple regression, to try to predict what "treatment" should be assigned to each student.

Student-Program Interface

This includes stimulus display, responses, and response records. Observe that responses include "trial responses" the student may make covertly before he commits himself to an observable response.

The problems here are to identify the stimulus displays, the response structures, and the response records that will be required. There will be many questions to be answered, and it is probable that work on this part of the specifications should be deferred until other parts of the system

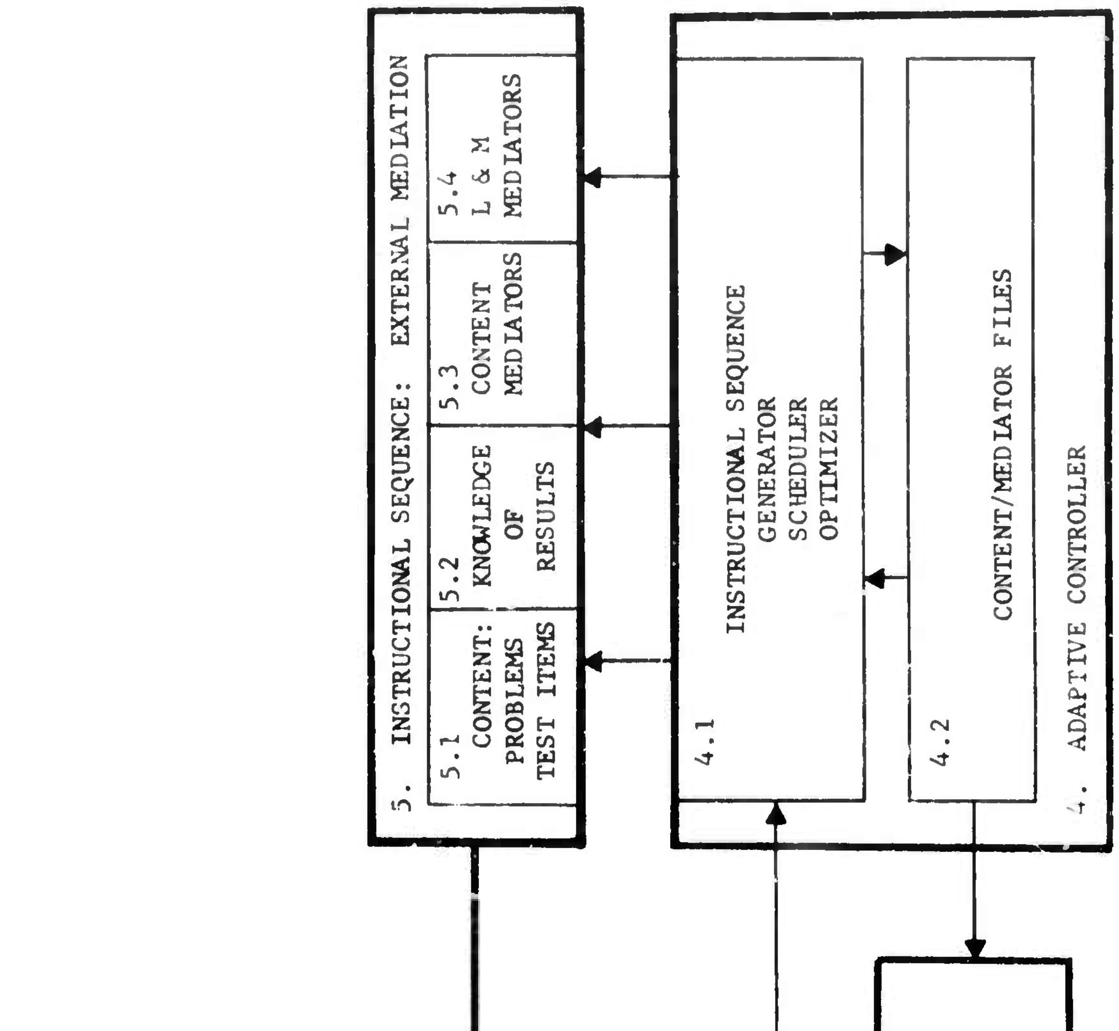


Figure 2. Outline of Major Elements in CAI

have been analyzed. For example, we would not know what to record about responses until we know what we want to do with these data, which depends on the type of adaptive controller used, and the type of statistical analysis that is required.

Student Data

This includes the student sufficient history computer and student records. The latter would include processed response records plus other information about the student; e.g., intelligence test scores.

A word or two about the concept of a student sufficient history computer is in order. The concept of a sufficient history was discussed by Atkinson and Paulson (1972). They stated that, "An index summarizing the information in a student's response protocol is a sufficient history if any additional information from the protocol would be redundant in the determination of the student's state of learning. The concept is analagous to a sufficient statistic..." We add the word "computer" to indicate not a piece of hardware, but software required to compute the sufficient history from response records and student records. For example, a Markov decision model we are developing (Wollmer, 1973) for adaptive control over the instructional sequence requires, for a student's sufficient history, records of the number of trials the student took at each level of a hierarchically organized course, the number of successful trials at each level, and the time spent in a level. From these data, collected from a preliminary sample of students, the model allows the computation of optimal patterns of successful trials to require of the next student in order to minimize time to complete the course. Each student's sufficient history becomes part of the information used in the optimization for the next student.

The same remarks about priorities of analysis apply here: What constitutes a student sufficient history will be determined by what is to be done with these data, which will depend on who will use the data and what they will be used for. Several different sufficient histories may be needed.

Adaptive Controller

This includes content/mediator files, essentially storage of course materials, an instructional sequence optimizer, an instructional sequence scheduler, and an instructional content generator. These three instructional operators require some explanation. An optimizer would be some method for improving the effectiveness of the CAI system, by optimizing learning rates under some set of constraints. Atkinson and Paulson (1972) have described general procedures for going about doing this.

The optimizer would identify an optimal instructional sequence for each individual student. The composition and sequencing of instruction is the principal way, if not the only way, to influence learning and retention in the CAI system. The optimizer would control an instructional sequence scheduler. If no optimizer is used, there still must be a scheduler. Instruction must be sequenced by some mechanism and that mechanism must be described in the specifications for the CAI system.

It is not yet clear, at this point in the technology, how general an optimizer-scheduler can be made. Atkinson (Atkinson and Paulson, 1972) has combined several different techniques for adaptive control over the sequencing of the strands in his reading program. It seems likely that this will be the case for other programs. General mathematical techniques will be represented in algorithms that must deal with highly specific data, learning models, and, possibly, constraints.

The instructional content generator could be either of two types: software that "makes up" the instruction on the spot from simple descriptive data, as in the TASKTEACH programs, or a team of instructors who write programmed instruction frames, and generate the entire sequence ahead of time.

Unfortunately, although writing CAI frames is a most time-consuming and expensive process, the range of applications in which the instructional content can be dynamically generated by a computer program currently is quite narrow. In the drill and practice mode it is easy to generate a series of math problems by changing the values of variables in a few standard problem structures. It is considerably more difficult, but still quite possible as we have shown, to generate the instructional sequence for a troubleshooting problem step-by-step, while the problem is being solved by a student, and to do this for troubleshooting a wide variety of devices. The only restriction on the range of devices is that certain of their characteristics and certain relationships in them be representable in a standard data-structure. It is possible as we also have demonstrated, (Rigney, et al., 1972a), to represent the structure of tasks in simple data-structures and to provide a general program to teach students how to perform a variety of tasks, either allowing them to compose their own individual instructional sequences, or placing the instructional sequence under program control. But it is not yet possible, so far as we know, to create a program to do task analysis or to generate the surface structure of specific tasks from a general, or primitive data structure analogous to Chomsky's (1964) deep structure. To do this, we would need a detailed model of how humans process information and organize their performance.

For the instructional content generator, the two major questions are: "What is the surface structure of the performance to be taught?" and "What are the content mediators that will be required in the training?" Content mediators are the operations and concepts in the material to be taught that bridge between stimulus and response. For example, in algebra, the mathematical operations and mathematical concepts required to solve a quadratic equation would be content mediators. We use this term to distinguish between these and learning and memory mediators, which are general operations the student may perform on broad categories of material to improve rate of learning and length of retention.

The content/mediator files in the adaptive controller store the material that is to be organized into an instructional sequence, or that is already organized into some "lesson" format. The way this material is stored and how much of it there is to be stored are important considerations. CAI requires a lot of storage, which usually takes the form of disc files. The random-access disc file is a common and almost indispensable feature of time-sharing systems. However, some CAI material may be stored more economically in peripheral devices in the form of slides or microfiches.

The optimizer, scheduler, and generator mechanisms in the adaptive controller tend today to be relatively crude procedures. These are areas that should receive much more intensive research and development, since they are at the core of the CAI system, and will determine its effectiveness.

Instructional Sequence

This sequence contains knowledge of results, subject-matter, and external mediators. It is the "input tape" to the student. We distinguish between knowledge of results as something the system provides for the student, and internal feedback as something the CNS provides, using information that comes to it both from the external and the internal environments. Knowledge of results provides some information for internal feedback, but this information may or may not be used. The knowledge of results provided by the system sometimes may be superfluous.

The two types of external mediators, content mediators and learning and memory mediators, have been defined earlier. Since most material is at some intermediate difficulty level in a roughly hierarchical structure, there usually are a number of different concepts, relations, and skills to be learned. These must somehow be organized into a sequence and presented to the student's very limited input system in serial order. Thus, the fundamental problem is how to schedule the instructional sequence in a way that will cover the different concepts, rules, and operations to be taught, and that will lead to optimum learning and retention rates.

As represented in the diagram, the instructional content would have been already composed by the generator and would be put into serial order by the scheduler, both in the adaptive controller. Thus, the instructional sequence is the output of operators in the adaptive controller.

Feedback Loops

The CAI system contains information channels that could be used to learn about the student and to learn about the adequacy of the instructional sequence, as well as to instruct and to provide knowledge of results to the student. Devising ways for the CAI system to learn about the student has turned out to be an extremely difficult thing to do. The Stanford CAI group (Atkinson and Paulson, 1972) was the first to tackle this and have done the most definitive work on this problem.

These feedback loops may be essentially open, because of long time delays, or essentially closed. For example, the feedback loop providing information about the adequacy of the instructional sequence usually is open. The very long time required to revise instructional materials usually means the revised version will be used on a different sample of students. For an extensive and provocative treatment of the topic of feedback in human behavior, see Powers (1973).

SECTION III

PROCEDURES FOR GENERATING SPECIFICATIONS FOR A PSO CAI SYSTEM

It is clear that the determination of what is to be taught requires identification and analysis of the tasks that the student is to learn to perform to some criteria of proficiency. This analysis will be the principal source of specifications. However, external objectives for the system also must be considered. Whoever is the customer for the system may want certain types of data from it which must be provided for in the specifications. For example, the customer is likely to want records of number of students put through the course, student scores, costs per student hour, etc. If the system is to be used for research objectives, or if there is to be an initial evaluation of it, these objectives are likely to have special requirements for data that usually exceed those for student sufficient histories for the customer.

These are, however, secondary considerations in comparison to the determining effects on the nature of the CAI system of the requirement to simulate essential aspects of the job performance-environment. Therefore, we shall tackle the task analysis problem first. We shall use three terms, surface structure, content mediators, and learning and memory (L & M) mediators, to divide the task analysis problem into three categories. We prefer these terms to training objectives and enabling objectives. Roughly speaking, the surface structure of a task performance will be the (mostly) observable actions of the performer, performing in a specific situation on a specific interface. Thus, the action on the front-panel controls and indicators required to tune the AN/URC-32 transmitter would be classified as surface structure. What the student has to know in order to tune this and other similar transmitters would be classified as content mediators. When he has learned enough of these, he can use them to generate the surface structure. How the student goes about learning and remembering content mediators are the functions of L & M mediators which the student happens to know how to use already or which are incorporated in the instructional sequence. It is possible, we believe, to teach learning and memory strategies to students.

As a consequence of learning, the student develops his own deep structure to guide him in performing without instructional assistance. The structure of this personal deep structure is unknown, of course. Nevertheless, we can, by using instructional techniques, assist students in developing deep structures. Our problem here is to develop procedures for identifying the implications of surface structures, content mediators, and L & M mediators for the six groups of elements in a CAI system, as diagrammed in Figure 2. This requires that we become concerned with the categorization of features as well as with specific details of these features.

PSO CAI is concerned with giving personnel drill and practice in performing technical jobs in which the boundaries of the performance are well-defined and performance cycles through well-defined episodes. This is characteristic of most operator and maintainer jobs. These jobs also may be embedded in an organization or system which requires the coordinated performances of many individuals and which controls the time and place of performance. The episodic, or cyclical nature of critical performance requirements can be diagrammed as in Figure 3.

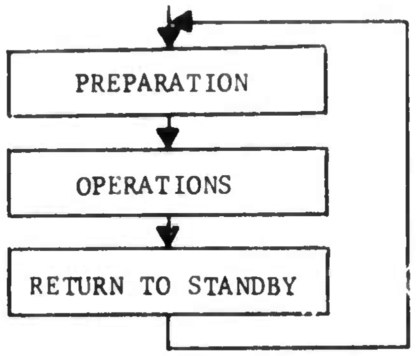


Figure 3. Overall Performance Cycle

Our principal concern in PSO CAI is with the operations part of this cycle, although in pre-CAI days we thoroughly analyzed the entire cycle for some CIC (Bryan, et al., 1956a,b) and maintenance (Rigney and Bond, 1964) jobs. The operations part of this cycle can be analyzed (as could the other two parts as well) into more and more detail, in terms of goal-action hierarchies, as in Figure 4.

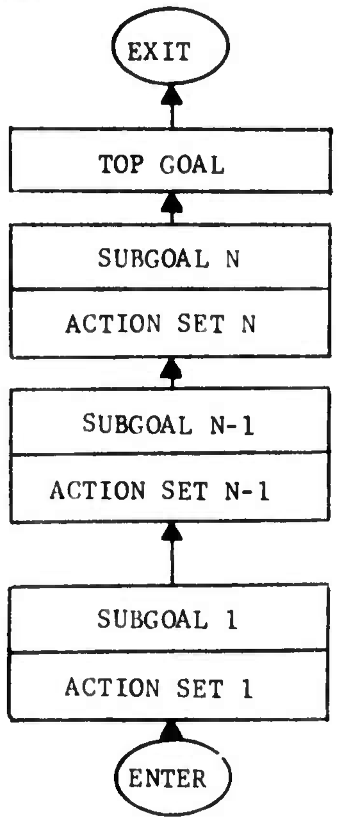


Figure 4. Performance as a Goal-Action Hierarchy

The structure of the performance can be analyzed in this way down to the industrial engineer's time and motion units (Rigney and Towne, 1969) if that is desired.

Categorization of Surface Structure Features

It is clear that surface structure features are major determinants of hardware design, particularly of the type of student terminal that would be required, since the terminal is the primary substitute for the performer performance-environment interface.

For the purposes of generating specifications for a CAI application, surface structure features of tasks can be categorized on the basis of interface characteristics with which the performer interacts. He receives information from various types of sources in the environment and he operates upon various types of objects in the environment. These may be characterized in a preliminary way, as outlined below.

1. Types of Displays in the Performance Environment
   - 1.1 Man-Machine Interfaces--Specific Displays
     - CRT Graphics
     - CRT Alphanumerics
     - Front Panels
   - 1.2 Extended Memories
     - Text
     - Diagrams
   - 1.3 General Environment
     - Visual
     - Auditory
     - Touch
   - 1.4 Static or Interactive

2. Responses in the Performance Environment
   - 2.1 Locus
     - 2.1.1 Man-Machine Interface
       - Discrete Controls
       - Continuous Controls
       - Tracking Controls
     - 2.1.2 Graphics Surface
       - Writing
       - Pointing
       - Drawing
     - 2.1.3 General Environment
   - 2.2 Type
     - 2.2.1 Reaching/Manipulative
     - 2.2.2 Locomotor

3. Special Lexicon
   - 3.1 Verbal Symbols
   - 3.2 Graphic General
   - 3.3 Graphic Mathematic
   - 3.4 Programming

Categorization of Content and L & M Mediators

In the ideal case, we would be able to describe suitable deep structures that students should possess. Consider, for example, the instructional time that is wasted in teaching students "theory" in training courses that turns out not to be needed or even useful for their job performance because we do not know the true nature of deep structures. Gagne (1970) pioneered thinking about the nature of deep structures in education and training. There is a fascinating and growing literature on the subject. This literature, up to about 1970, was reviewed in an earlier report (Rigney, 1971).

Subsequently, there have been striking developments in modeling the structure of long-term memory; e.g., Rumelhart, Lindsay, and Norman (1972), Anderson and Bower (1973), which are excellently summarized in Anderson and Bower (1973). This work is leading toward eventual simulation of human cognitive processes, which has enormously important implications for education and training. Eventually, we may be able to represent deep structures stored in long-term memory and to operate upon these to produce surface structures in ways characteristic of human learning and memory. An example of a propositional deep structure representation of a sentence, used in Anderson and Bower's model, is given below (Figure 5).

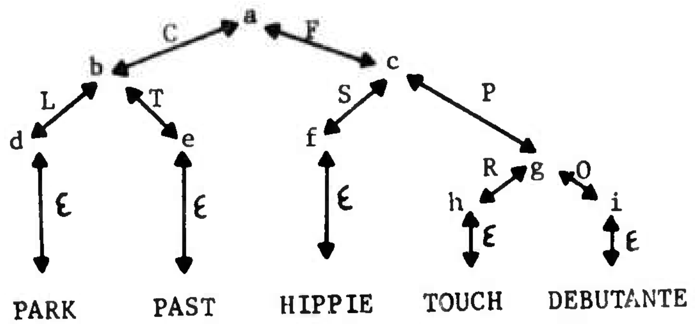


Figure 5. An Example of the Propositional Representation of the Sentence: "In a Park a Hippie Touched a Debutante". From Anderson and Bower (1973), page 139

C = Context	F = Fact	P = Predicate
L = Location	S = Subject	R = Relation
T = Time		O = Object

ε = Set membership; a, b, ... i = Nodes

(Nodes represent ideas and the links represent relations or associations between these ideas)

Language is a form of performance. Written or spoken sentences are said, since Chomsky (1964) originated the concept, to be the surface structure of language. This is what is seen, heard, or spoken. But this surface structure can vary a great deal and still convey the same meaning.

Sentences can be paraphrased without destroying the essential information they communicate. There is, Chomsky said, a deep structure that conveys the meaning, and that is the basis for generating the surface structure of language.

An analogy can be drawn between other forms of human performance and language performance. Beginners must stumble along or be led by the hand until they learn a deep structure that will let them generate the desired performance. PSO CAI is designed to give them practice in generating this performance. It is designed to be extended back into the structure of content mediators by providing drills in subskills which underly the surface performance. It gives the student the opportunity to organize what he already knows and to identify what he needs to know to generate the surface structure of whatever performance is required for the job. We propose that there are powerful self-organizing processes in the central nervous system, that function in ways as yet very poorly known, to develop deep structures. These self-organizing processes have the opportunity to function during practice in performing surface structures. In drawing this analogy, we also should point out the obvious fact that language is used in learning to perform tasks. As a representational process, as a means for communicating instructions, as the basis for creating shared contexts which allow humans to "stand aside and look" at what they have done, are doing, or are going to do, language is a universal and indispensable learning tool. In fact, it is possible that the same deep structure in long-term memory, what Anderson and Bower (1973) call the "strategy-free component of memory" is the basis for all cognitively-controlled performance.

For the present, however, our objective must be the identification of important categories of content mediators embedded in the surface structures of the tasks to be taught, so that appropriate instructional sequences can be composed and scheduled, the appropriate subroutines can be put into the computer program to give drills in learning these content mediators; the appropriate stimulus displays can be designed, etc.

At this point in history, the analysis of task content mediators is very much an empirical matter. Some rather broad assumptions must be made about the common cultural skills and knowledge possessed by the target population of students. Only the content mediators which become apparent from the surface structure analysis should be examined. These could be divided into two groups; representations and operations. Representations are models of and abstractions from environmental features; concepts, objects, processes, and events, that students must understand to be able to perform. Operations are information-processing operations the student must be able to do mentally to be able to perform. They may be classified in many ways, as various "taxonomies;" e.g., Gagne (1965); Bloom, et al., (1956) testify.

Taxonomies like these would be useful for generating specifications for software for CAI only to the extent that they suggest the directions that more detailed analyses must take, or identify groups of operations that have been or could be implemented in computer programs. Otherwise, they are entirely too general.

These classifications and taxonomies tend to overlook the structures of operations. For example, there is a certain order of information-processing events characteristic of particular content mediators. Indeed, knowing the sequence of operations allows the student to predict "what to do next" and to program himself to do it. The analysis of content mediators must become concerned with the sequences of operations and with ways to represent these to the student, for these serial patterns are themselves content mediators. We have done a considerable amount of work on this problem. A computer program has been produced to teach students the serial structure of operations on man-machine interfaces (Rigney, et al., 1972a) which is a step in the direction of a "task parser" analogous to a sentence parser used in natural language processing.

The program, called CAPTIVE, operates on a list of action and goal statements organized by a numerical code that describes the syntax of the performance in terms of action and goal sequence constraints, goal set membership of actions, etc. The program can recognize when a student's performance is "grammatical", that is, when it follows the rules of syntax. It can take a string of actions apart somewhat like a sentence-parser, but it goes far beyond a parser in its ability to guide a student in learning to use the correct syntax.

Since the principal implications of content mediators for CAI system specifications are for computer programming, analysis must become quite detailed. The principal analytical tool to be used is the instructional-sequence flowchart. Ideally, this would make content mediators explicit along with the specification of surface structure considerations.

Learning and memory mediators are more or less "content-free" operations that the student could perform to improve his rate of learning and/or his length of retention. We say more or less content-free because it is unlikely that all L & M strategies useful for learning mathematics, for example, would be equally useful for comprehending and remembering narrative structures of history or literature.

Recently, there has been a revival of interest in the effects of imagery, and to a lesser extent in other learning strategies, stimulated in part by demonstrations of the startling effectiveness of these strategies in learning and remembering nonsense syllables (e.g., Bower, 1972; Prytulak, 1971).

It does seem important that L & M mediators should be incorporated in CAI systems, insofar as the experimental evidence suggests that they would be worthwhile. It is even more important that research on the effectiveness of these strategies be done in the context of CAI, since this is quite different from the usual laboratory setting.

It is time we stop categorizing human students with simpler laboratory animals like rats and pigeons and recognize that animal conditioning paradigms are insufficient bases for CAI. The mental operations that students perform while learning are crucially important determinants of how well students learn and remember. Our problem is to achieve a better understanding of what these mental operations are, and to gain enough control over them to increase the effectiveness of CAI.

For this reason, we believe L & M mediators should be included in planning, even though knowledge about what they are and how to use them outside of the experimental laboratory is still very scant. We include a brief summary of the research literature on imagery, an example of an L & M mediator, in the following paragraphs.

The image-evoking (I) value or concreteness of stimulus material is positively correlated with the learnability of that material. Recall and recognition are greater for material of high I value. (Paivio and Madigan, 1968, 1970; Begg and Paivio, 1969; Paivio, 1969; Paivio and Csapo, 1969; Paivio and Yuille, 1969; Paivio, Yuille, and Rogers, 1969; Paivio and Rowe, 1970; Paivio and Smythe, 1971; Anderson, 1973; Craig, 1973; Griffith and Johnson, 1973; and Montague and Carter, 1973).

Experimental manipulations of imagery affect learning. Two types of variables have been studied. (1) Instructions (or prior set) to form interactive images increases recall and recognition of the to-be-learned material. (Paivio and Yuille, 1967, 1969; Bugelski, 1968; Bugelski, Kidd, and Segmen, 1968; Bower, 1970; Anderson, 1971; Anderson and Hidde, 1971; Bower, 1972; Anderson and Bower, 1973; Griffith and Johnston, 1973; and Levin, 1973.) (2) Providing mediating pictures facilitates learning. (Davidson, 1964; Reese, 1965; Rohwer, Lynch, Levin, and Suzuki, 1967; Lippman and Shanahan, 1973; and Nelson and Brooks, 1973.)

Individual differences in effective imagery potential affects learning. Individuals classified as high in ability to image recalled or recognized more than those rated low in imaging ability. (Ernst and Paivio, 1969, 1971; Sheehan, 1966, 1971; Paivio and Okovita, 1971; and Marks, 1972.)

Implications of Task Analyses for CAI System Specifications

The outlines of a hypothetical mechanism for generating CAI system specifications are illustrated in Figure 6. Each different feature in the task structures might be "run through" the impact analyzer to see if that feature would have an impact on any of the six CAI system elements, considering each in turn. If there would be no impact on an element, the next element would be considered for that feature, and so on, until an element was found that was impacted by the feature. In that case, the hardware implications analyzer would be called to analyze the implications of the feature for that element. There would be sub-analyzers for major hardware components; memory, CPU, and terminal sub-analyzers are indicated in the figure. The outputs of the implications analyzers would go to the specifications generators, which, in the case of hardware, would look for matches between requirements and available items, and could list alternatives. For example, animated, interactive graphics might be desirable, if not required, to illustrate processes to the student, and to engage him in exploring these processes, say the solid-state physics that are the basis for transistors. Two types of graphics terminals might be listed in a "catalog" of hardware items known to be on the market. This catalog would be in the form of a data base which the terminals specification generator would search. Suppose one of the terminals provided a light pen with very accurate X,Y coordinate discrimination and the other provided a touch panel with crude

X,Y coordinate discrimination. The terminal specification generator would need data describing the maximum accuracy of X,Y discrimination ever required in the student-program interface by the particular application under analysis. If this type of accuracy is unimportant, the generator might select the terminal with the touch panel. The same procedure could be followed for software, except that the mechanisms in the implications analyzers and the specifications generators would differ. As n features were each run through six CAI system elements, it is likely that many overlapping or redundant specifications would be generated. These would be collected and reduced to the shortest complete list. It can be argued that Figure 6 illustrates the general operations involved in generating CAI specifications from task analysis information, but, in practice, the analyst will find that he need not go through every single step that is implied by this diagram. There already are general specifications for PSO CAI that serve to structure and to simplify the task of generating specifications for a particular application. PSO CAI is based on certain hardware capabilities that can be assumed to be available for any application:

1. Interactive, dynamic (animated) computer graphics.

2. Input-output interfacing for computer-controlled front-panel simulators, voice synthesizers, and image projectors.

3. Light pen (or the equivalent) and keyboard for student response inputs. It also must be possible to interface with the computer special response-input devices, e.g., control sticks, and analog and digital controls and switches on man-machine interfaces.

4. Computer-controlled peripherals for producing hard copy and for bulk storage. (The most economical hardware currently available are teletypes for hard-copy and floppy disks for storage.)

5. CPU cycle-time sufficiently fast to permit real-time simulation involving accepting and responding to student inputs, and updating multiple variables within one "frame" or one refresh cycle.

6. Complete CAI system integrated into each terminal.

All of these capabilities will be in the "CAI testbed" hardware system as illustrated in Figure 7, being developed under Office of Naval Research funding.

The problem for generating hardware specifications for specific applications, given the above general capabilities, is to identify the additional, special man-machine interfaces these applications would require, since only these would have to be designed for each application. Surface-structure features would be used to identify these special man-machine interface requirements.

PSO CAI also is based on a certain general structure for the part of the CAI system, which is most of it, that is implemented by software; i.e., by computer programs. The software capabilities generally required for PSO CAI can be characterized as follows:

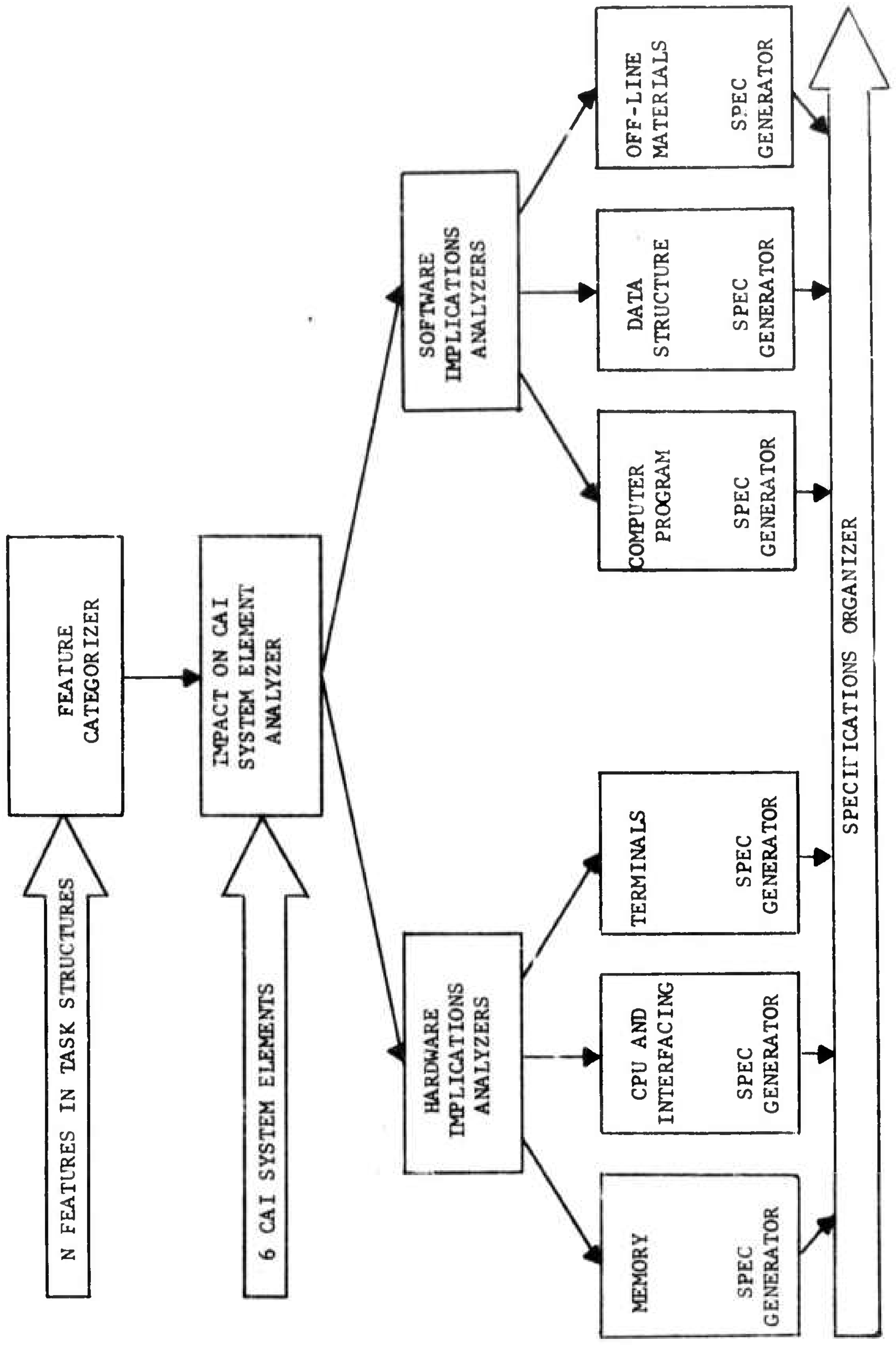

N FEATURES IN TASK STRUCTURES
FEATURE CATEGORIZER
6 CAI SYSTEM ELEMENTS
IMPACT ON CAI SYSTEM ELEMENT ANALYZER
HARDWARE IMPLICATIONS ANALYZERS
SOFTWARE IMPLICATIONS ANALYZERS
MEMORY SPEC GENERATOR
CPU AND INTERFACING SPEC GENERATOR
TERMINALS SPEC GENERATOR
COMPUTER PROGRAM SPEC GENERATOR
DATA STRUCTURE SPEC GENERATOR
OFF-LINE MATERIALS SPEC GENERATOR
SPECIFICATIONS ORGANIZER

Figure 6. Tentative Outline of Procedure for Generating CAI System Specifications from an Analysis of the Performance to be Taught

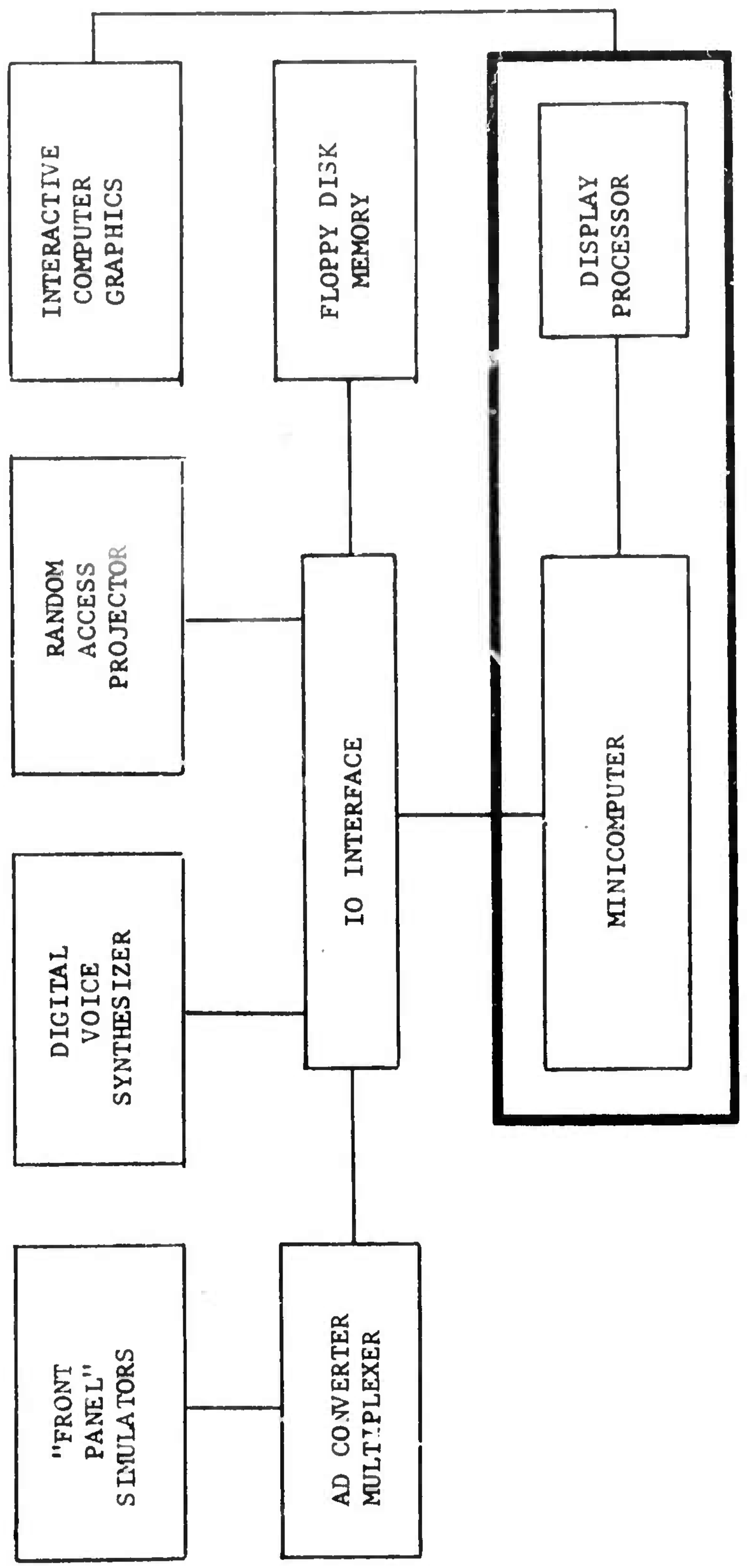


Figure 7. Block Diagram of Hands-on Training CAI Testbed Hardware

1. Unique interaction with each student, via dynamic computer graphics and/or special student-program interfaces. The computer program must accept the student's input, calculate the values of variables required to respond to this input, and display these values in suitable format to the student.

2. Continuous updating of the simulated performance environment that is the subject of the training. (In the case of the RIO trainer, which was a real-time problem, this updating occurred every tenth of second and every time a student responded.)

3. Continuous response-by-response tracking of student progress. This is necessary to generate the interaction with the student, to provide knowledge of results, and to provide data for student records.

4. Instructional functions that can be placed either under student or under program control.

5. Response-sensitive adaptive control over the instructional sequence. Currently, trials-to-criterion logic is used. (The possibilities for using dynamic programming are under investigation.)

6. Multiple response-variable recording and analysis. The use of multiple dependent variables provides needed information about what was going on during learning.

7. Generation of the instructional sequence from program logic operating on simple data-structures.

There are several techniques that might be considered for reducing the amount of programming required for new applications of PSO CAI. Standardization of operations described for the five CAI system elements (see Figure 2) might allow some of the same subroutines to be used across different applications by passing the specific values of variables to these subroutines.

Standardization would necessarily extend to conventions for data-structures as well. Programs and the data-structures on which they operate are so closely interrelated as to require joint design. Standardization also would include a system executive routine to manage the other programs. Typically such an executive program cycles through a list of "things to be done" within some time frame, accepts certain interrupts from the external world, calls subroutines to service the interrupts, and returns to the cycle. However, it is likely that this executive would be a fairly short program, and thus would not contribute a great deal to reducing the programming burden.

Modularization of programs would make it possible to separate subroutines that perform general operations from programs that perform operations specific to an application. Modularization is a fundamental requirement for any system of programs; for convenient revisions, for growth, for using techniques such as overlay or virtual memory, and for more effective documentation.

Writing interpreters for instructors to use for constructing computer graphics and data-structures, both of which must be specific to an application, would allow instructors to do this relatively quickly. The interpreter takes instructions written in a higher-level language (source code) and translates (interprets) them into machine language (object code) as they are executed by the computer. Thus, an interpreter provides a more convenient and faster method for constructing data-structures, which would be specific to the application.

We have produced an interpreter for computer graphics that allows the instructor to construct figures by using function keys on the graphics terminal keyboard. The interpretive program interprets these function keys by writing subroutines to display the figures. An editor is included in the interpreter to allow changes and corrections to be made to the figure while it is being composed. The resulting display subroutines are stored in core, where they may be accessed by other programs (Rigney and Towne, in press).

We also have produced an interpreter for constructing data-structures for the TASKTEACH troubleshooting programs. The instructor need only identify and list inputs and outputs of each circuit in a device, using a numerical code. The interpreter then will reconstruct the entire network and generate from it certain types of relational information.

This experience leads us to conclude that it is extremely worthwhile to develop interpreters for CAI, since they save an enormous amount of man hours otherwise necessary. We see no reasons why interpreters could not be developed for other areas in CAI as well.

Finally, a compiler for some standard programming language; e.g., Fortran IV or Basic, would, of course, expedite producing all the computer programs, at the expense of possibly using a little more memory for the object code. At the present time, only an assembler/editor is available for the minicomputer used in the PSO CAI hardware system.

The big question is just how far generalization techniques can be pushed to develop what might be called a general operating system for PSO CAI. The successes we have had in doing this for limited areas have also taught us that this is far from a trivial problem. Nevertheless, the enormous amount of labor required to produce instructional programs for the alternative, "ad hoc frame-oriented" (Carbonell, 1972) CAI, is, we believe, unacceptable in the long run. The problem will be given more intensive consideration later. Meanwhile, in the following sections we will turn our attention to procedures for generating software specifications.

Examples of Procedures for Generating Software Specifications

In the case of specifications for software, it has been our experience that producing an instructional sequence flowchart serves to force the analysis in the right directions, and results in a means for communicating the basic structure of the instructional sequence to the computer programmers. This is, we hasten to add, an iterative process. When computer programming begins, it uncovers loose ends that were left unresolved or that were overlooked that now must be finished. We emphasize that the instructional sequence flowchart is the primary outcome of the task analysis, and is the principal medium for defining and communicating specifications. It will be the source of specifications for stimulus and response features, content mediators, L & M mediators, knowledge of results, and for their sequential organization.

The task analysis to identify surface structures and content mediators, and to select L & M mediators, must be summarized in this instructional flowchart. This is why we find that the products of the current fad in military training for painstakingly writing out long lists of behavioral objectives are not very useful. We surveyed this field recently (see Appendix B) and concluded, reluctantly, that these analyses are simply too general, leaving out details that are essential. It is necessary to do an analysis that will produce the exact information needed for this purpose. An example of a reasonably detailed task analysis for the RIO's job in air intercepts is reproduced below. This example includes the analysis only for the first of four major goals during the intercept (1) establish collision course, (2) establish displacement distance, (3) primary attack, and (4) reattack. However, it illustrates about the right level of detail for this type of analysis.

Activity List; Goal 1: Closure (Collision Course)

<u>GOAL 1</u>: Establish a "best closure" on any designated target. (Co-speed)	
Action 1: Receive Bogey data from GCI Controller -- Verbal (Radio). a. Bogey Bearing (BB) Degrees True b. Bogey Range (BR) miles c. Bogey Heading (BH) Degrees True	1. Alphanumeric display 2. Additional Data: Fighter Heading (FH)
Action 2: Determine Reciprocal of Bogey Heading (BHR) to fix initial values of the intercept triangle (Figure 1). a. If $BH < 180^\circ$ True $BHR = BH + 180^\circ$ b. If $BH > 180^\circ$ True $BHR = BH - 180^\circ$	Errors: miscomputes BHR
Action 3: From BHR and BB determine Target Aspect Angle (TA). a. If BB right of BHR TA right b. If BB left of BHR TA left c. TA = Difference in degrees between BHR and BB	Errors: 1. Wrong value of TA 2. TA determined in wrong direction

Action 4:
From Fighter Heading (FH) and Bogey Bearing (BB) determine Angle Off (AO). Label AO "right" or "left."
a. If BB < FH; AO "left"
b. If BB > FH; AO "right"

Angle Off in degrees: the bearing of the Bogey relative to the longitudinal axis of the Fighter, measured in degrees left or right of the nose.

Errors:
1. Wrong value of AO
2. Wrong direction of AO

Action 5:
Fighter position in relation to Bogey flight path.
a. If TA and AO <u>both</u> "left" or "right" fighter is <u>across</u> Bogey flight path
b. If TA and AO different, fighter closing on Bogey flight path

Fighter across Bogey flight path (heading away from Bogey), separation between the two A/C is increasing.

Errors: If "labels" (right/left) of TA and AO erroneously determined, fighter position error, could cause turns in wrong direction.

Action 6:
Determine aircraft (fighter) "best" initial closure course; collision course (CC)
a. Collision Course = that fighter heading where TA = AO
b. Find difference between TA and AO

"Best" initial closure course is the "collision course," i.e., that fighter heading that would cause the two A/C to collide if at same altitude.

Errors: determines wrong value of difference between TA/AO.

Action 7:
Turn fighter to Collision Course.
a. If AO "left," turn left
b. If AO "right," turn right

Errors:
1. Turns wrong direction
2. Turns wrong No. of degrees

Proper directive terminology:
"Turn left/right (hard, easy), XX to XXX"
XX: No. of degrees of turn, Exp. 20, 30, etc.
XXX: Fighter Heading in degrees, Exp. 350, 270, etc.

Action 8:
Determine number of Degrees To Go to Bogey Heading
a. DTG = Difference FH and BH

Degrees To Go (DTG): the number of degrees of difference between Fighter Heading and Bogey Heading; the numbers of degrees Fighter must turn (left or right) to parallel Bogey Heading.
(This action could also be performed <u>prior to</u> turn to CC and would be

	determined again after turn to CC)
Action 9: With Fighter steady on new (Collision) Heading: a. Give AREO Report b. Give DTG c. Test for good CC establishment: a collision course is good if BB/AO holds a constant value.	AREO: A report to pilot to provide information on Bogey. A = Angle Off R = Range E = Bogey Elevation in feet above/below Flight Altitude (if known) O = Overtake (closure) if known Test for CC: 1. Bogey not yet acquired on fighter radar: a. Receives Bogey Bearing from GCI, determine AO. (1) If AO increasing/decreasing, CC not adequate. Errors: 1. AREO Report errors in value or terminology 2. DTG value 3. Fails to detect poor CC establishment
Action 10: Corrects collision course	Repeats Action 6, 7 If bearing change small may shift to Goal 2, and acquire Bogey on radar.

As we pointed out above, producing detailed specifications for computer programs is an iterative process. The first flowcharts will be rather general, but they will be useful for sustaining the dialog between the instructional technologist and the programmer. The main requirement is to put down the flow of the different events that are planned for the instructional sequence, in terms of major events and decision boxes. Then, this crude model can be analyzed into final detail. An example of a preliminary flowchart for one phase of the RIO's job in air intercepts is given in Figure 8. During this process, many revisions will be necessary; since some planned displays or operations may be infeasible to program, or may be in error, or better ways of accomplishing certain objectives may be thought of during discussions. Thus, the instruction flowchart is a medium of communication between two different technologists, and as such, should remain fluid. It would be a serious mistake to try to produce one, grand, detailed, final instructional flowchart before "calling in" the programmers. Ultimately, something like the flowchart reproduced in Figure 9 will evolve. By this time, it will be more like documentation of what has been done than a plan for what is to be done. It will reflect the interaction between instructional technologist and programmer by containing many terms characteristically produced by programmers. Figure 9 reproduces the complete flowchart, at a rather general level, considering a programmer's needs, for the RIO intercept training computer program, as it was finally evolved.

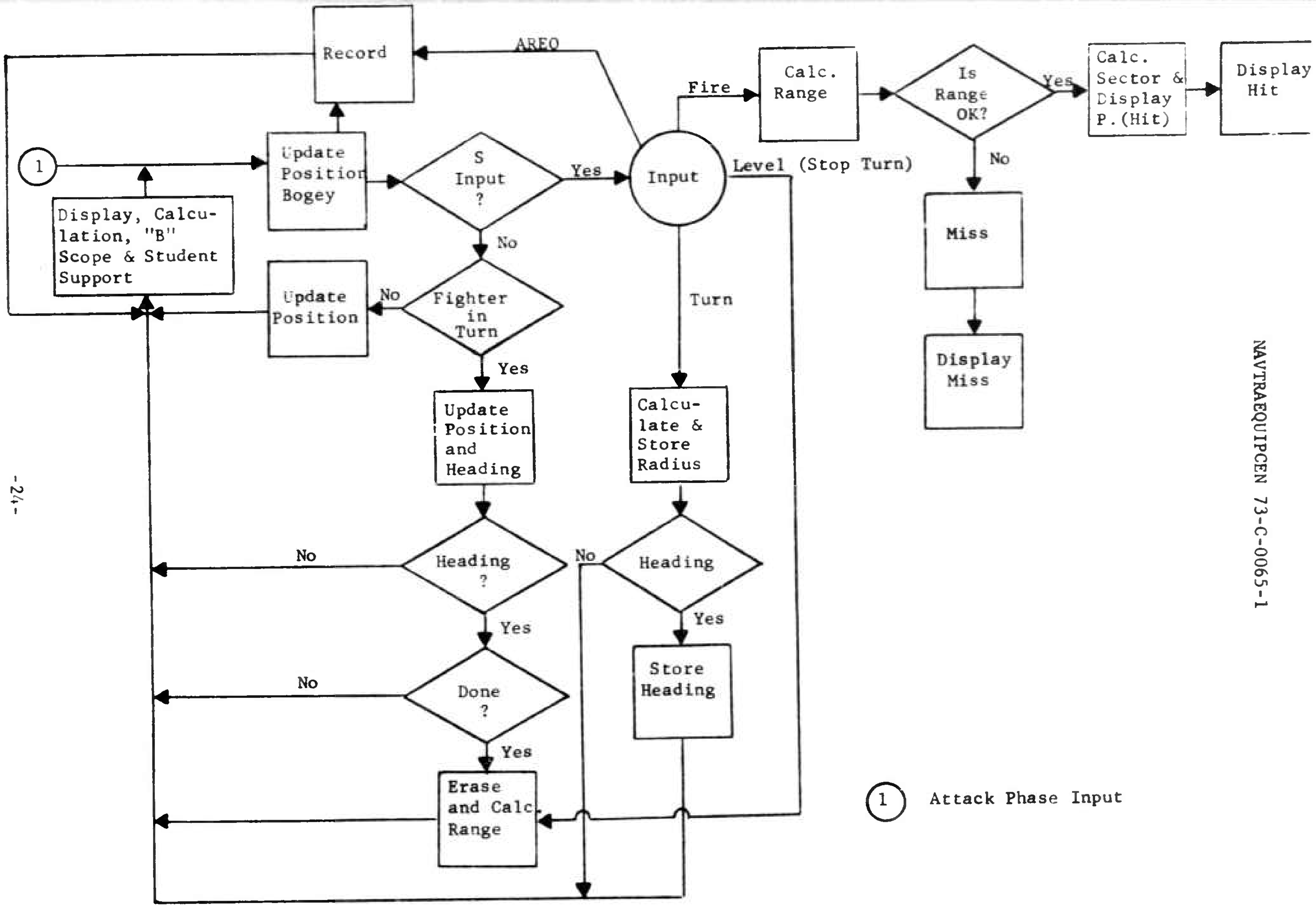
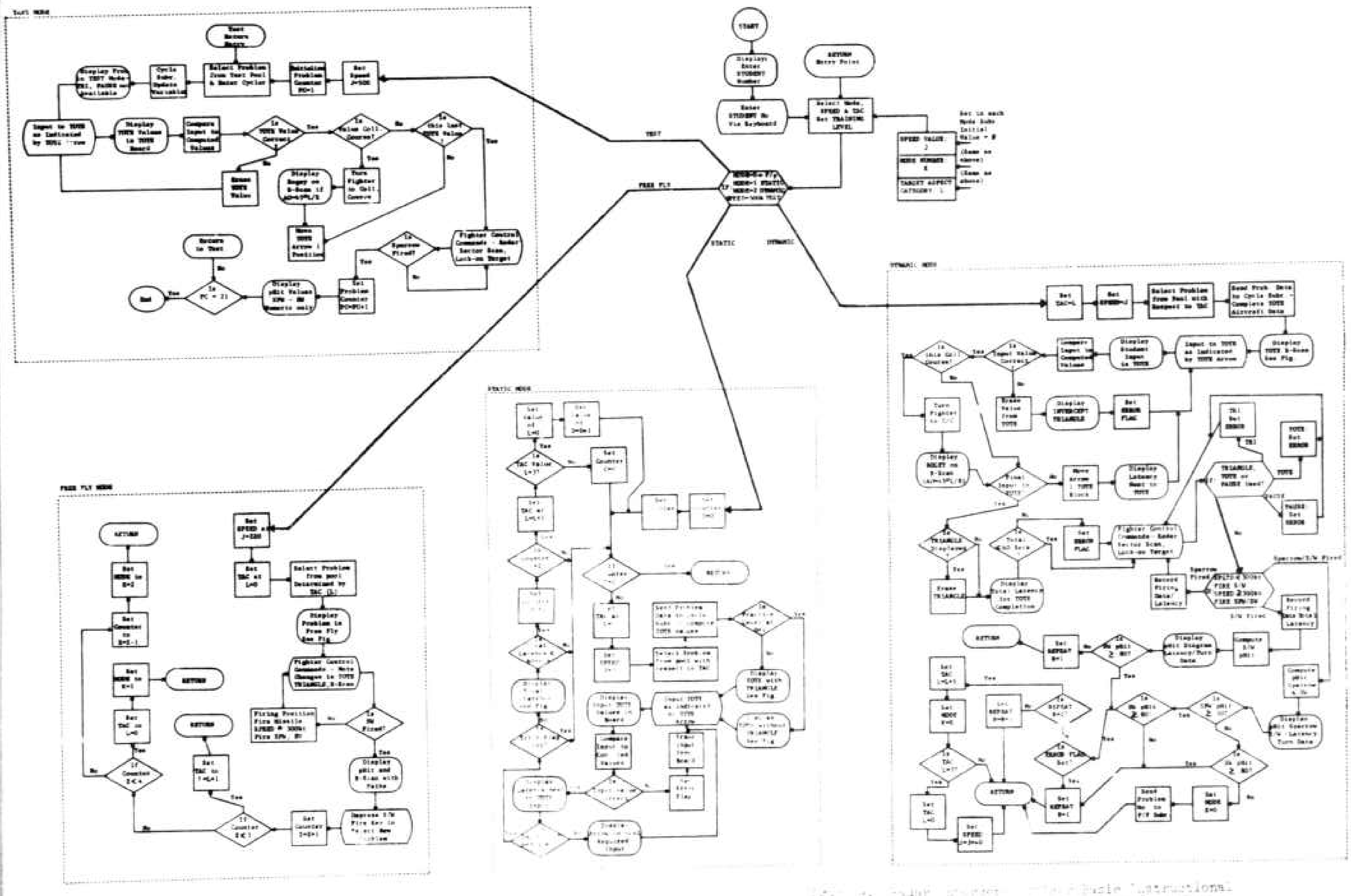


Figure 8. Preliminary Flowchart for Driving Interaction Cycle: Attack Phase, RIO Trainer


START
Display: Enter STUDENT Number
Enter STUDENT No. Via Keyboard
RETURN Entry Point
Select Mode, SPEED & TAC Set TRAINING LEVEL
SPEED VALUE 2
MODE NUMBER X
TARGET ASPECT CATEGORY 1
Set to each Mode Subr. Initial Value = 0
(Same as above)
TEST
FREE FLY
STATIC
DYNAMIC
TEST MODE
Test Return Entry
Select Problem from Test Pool & Enter Cycler
Initialize Problem Counter PC=1
Set Speed J=SOX
Cycle Subr. Update Variables
Display Prob. in TEST Mode - TRI, PAIRS not available
Input to TOTE as Indicated by TOTE Arrow
Display TOTE Values in TOTE Board
Compare Input to Computed Values
Is TOTE Value Correct?
Is Value Coll. Course?
Is this Last TOTE Value?
Erase TOTE Value
Display Bogey on B-Scan if ρ&Δ=45°L/R
Turn Fighter to Coll. Course
Move TOTE Arrow 1 Position
Fighter Control Commands Radar Sector Scan, Lock-on Target
Is Sparrow Fired?
Set Problem Counter PC=PC+1
Display pHit Values SPW - SW Moments only
Is PC = 21
Return to Test
End
Yes
No
FREE FLY MODE
Set SPEED at J=...
Set TAC at L=0
Select Problem from pool Determined by TAC (L)
Display Problem in Free Fly See Fig.
Fighter Control Commands - Note Changes to TOTE TRIANGLE, B-Scan
Is SW Fired?
Firing Position Fire Missile SPEED ≥ 300kt Fire SPW, SW
Display pHit and B-Scan with Paths
Depress S/W Fire Key to Select New Problem
Set Counter I=I+1
If Counter I ≤ 1
Set TAC to L=L+1
RETURN
If Counter I ≤ 4
Set TAC to L=0
Set MODE to K=1
Set Counter to I=I-1
Set MODE to K=2
STATIC MODE
DYNAMIC MODE
Set TAC=L
Set SPEED=J
Select Problem from Pool with Respect to TAC
Read Prob. Data to Cycle Subr. - Complete TOTE Aircraft Data
Display TOTE B-Scan See Fig.
Input to TOTE as Indicated by TOTE Arrow
Display Student Input in TOTE
Compare Input to Computed Values
Is Input Value Correct?
Is this Coll. Course?
Erase Value from TOTE
Display INTERCEPT TRIANGLE
Set ERROR FLAG
Turn Fighter to C/C
Display BOGEY on B-Scan
Final Input to TOTE?
Move Arrow 1 TOTE Block
Display Latency Next to TOTE
Is TRIANGLE Displayed?
Erase TRIANGLE
Is Total Err Zero
Display Total Latency for TOTE Completion
Set ERROR FLAG
Fighter Control Commands Radar Sector Scan, Lock-on Target
TRI Set ERROR
TRIANGLE, TOTE or PAIRS Used?
TOTE Set ERROR
PAIRS Set ERROR
Sparrow Fired
SPEED ≤ 300kt FIRE S/W SPEED ≥ 300kt FIRE SPW/SW
Record Firing Data/Latency
Sparrow/S/W Fired
S/W Fired
Record Firing Data S&B Latency
Compute S/W pHit
Display pHit Diagram Latency/Turn Data
Is pHit ≥ 80?
Set REPEAT R=1
RETURN
Compute pHit Sparrow & SW
Display pHit Sparrow S/W Latency Turn Data
Is SW pHit ≥ 80?
Is Sw pHit ≥ 80?
Is REPEAT R=1?
Is ERROR FLAG Set?
Set REPEAT R=0
Set REPEAT R=1
Set MODE K=0
Read Problem No. to F/F Subr.
Set TAC L=L+1
Set MODE K=0
Is TAC L=3?
Set TAC L=0
Set SPEED J=J+1
RETURN

[illegible] Radar [illegible] Basic Instructional [illegible]

<u>Legend of Variables</u>:

Speed = J:	Varies over the range 220 to 500 knots. Initialized at 220 kts, increases by increments of 40 kts determined by student performance. As criterion performance is achieved, speed is incremented and further practice is required until the 500 kts level is reached. The Test problems are performed at this speed level. At speeds of 300 knots and above both Sparrow and Sidewinder are fired.
Mode Number = K:	Utilized to branch the student through the primary practice modes: Mode Ø = Free Fly, Mode 1 = Static Practice, Mode 2 = Dynamic Practice.
Target Aspect Catagory (TAC) = L:	Utilized to select problems of a given TAC. Since Target Aspect is a variable which determines the degree of maneuvering and technique required in an intercept, TAC is arbitrarily divided into Low, Medium, and High categories. The instructional strategy requires that the student perform an equal number of intercepts in each category to meet performance criteria.
TOTE:	The displayed list of computational variables that the student must determine and input to the program prior to commencing an intercept. Static Mode is the practice module which develops speed and accuracy in determining correct TOTE values. Depression of the TOTE Key during and intercept will cause display of current values of all intercept variables.
TRI:	Intercept Triangle: A plan view of the basic triangle formed by the intersection of Bogey course line, Fighter Course line, and the line of Bogey Bearing from the Fighter. In Static Mode the triangle is static (the fighter and Bogey positions are fixed). In Dynamic Mode the triangle is updated in real time representing the relative positions of the Bogey and Fighter as the intercept progresses. The triangle is displayed automatically if a TOTE error occurs or on demand when the TRI Key is held depressed.
Pause:	A student-controlled program function. Depressing the Pause Key effectively "freezes" the problem allowing the student to evaluate elements of the intercept. A second depression continues the problem from the instant of pausing.
pHit:	Hit probability - a numerical score determined for both the Sparrow and Sidewinder missiles by arbitrary algorithm. This number is an index of the student's proximity to optimal range angle off the target, and turn condition at the time of weapon launch. A hit probability of .80 for the Sidewinder and .50 for the Sparrow must be achieved to attain criterion performance.
Latency:	The time required to perform an intercept function. Latency is measured between responses on TOTE, for the overall completion of the TOTE list (total latency). A latency of 60 seconds is established as the criterion for TOTE performance.

Student-Program Interface: Stimulus Display. For computer graphics, features for the stimulus display must be specified in terms of what is to be unchanging and what is to change on the display. The course starts with a certain type of display. Features of this change in relation to the student's responses, under the control of the Instructional-Sequence Scheduler. What might be called the cross-sectional and the longitudinal features of the stimulus display must be identified. This is best done in terms of diagrams of all the different display configurations, in which each feature is drawn to scale as it is to appear, and is integrated with the instructional flowchart that shows details of the sequencing logic for the different displays.

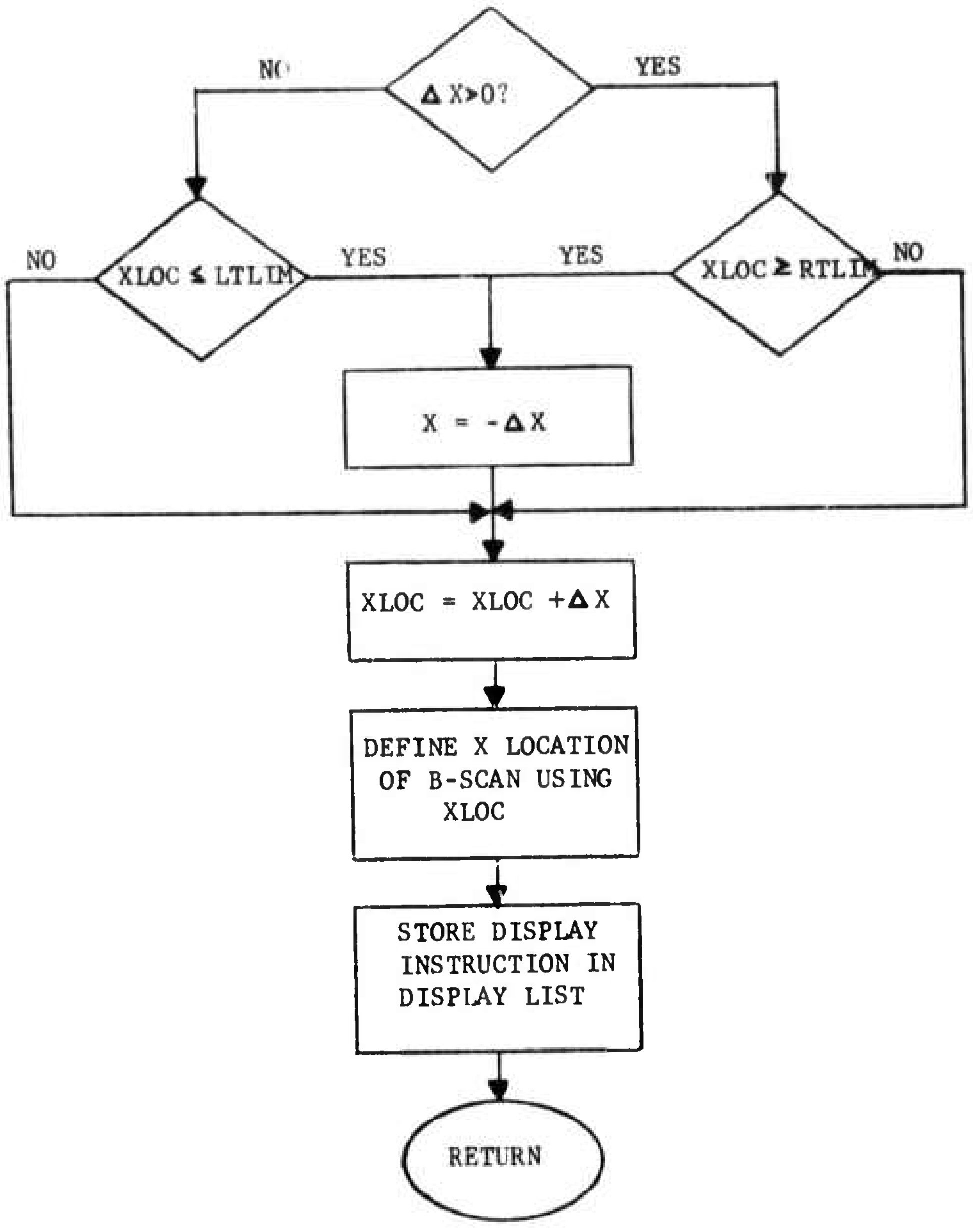
The programmer's flowcharts for computer-graphics programs can be constructed from these diagrams and the instructional flowcharts, although as always, several revisions of flowcharts and much verbal interaction between the instructional technologist and the computer programmer will be required. The display generator programs will necessarily interface with, or will be subroutines in the program for generating the interaction with the student, which accepts student inputs that will influence the composition of the displays. An example of a programmer's flowchart for simulating a radar B-scan graphically is given in Figure 10. It will be seen that this is at a finer level of detail than the instructional flowchart reproduced in Figure 9. This will always be the case.

Computer graphics are drawn by subroutines which run in a display processor computer, and which are made up of special display instructions. Computer graphics typically require a lot of core, since a great many computer words are needed to draw even a moderately complicated figure.

Specifications for special interfaces; e.g., front panels of operational equipment, would be determined from the instructional flowchart in a manner similar to the derivation of specifications for computer graphics. Each display and each control on the front panel must be identified and its role in the instructional sequence must be described. This involves identifying different states of the device that are established by different configurations of front panel controls, and describing these states and conditions in a data structure somewhat analogous to the data structure for computer graphics.

These special interfaces also require that appropriate hardware be designed for representing the interface and for attaching it to the computer. The IO interface and the A/D converter multiplexer, indicated in Figure 7, are examples of this type of hardware. This generally can be relatively simple, partly because humans respond at relatively low data-rates. A good bit of software is required to make this hardware work; transmission of information back and forth must be under program control.

Student-Program Interface: Response Recording. Student responses may be made via a keyboard, light-pen, or special control on a special man-machine panel. In all cases, the computer program looks in an input register to find the response information. Student inputs in raw or processed form will be used by the instructional sequence (IS) generator, scheduler, and optimizer. This IS generator uses them as part of the information needed for controlling the immediate student-program interaction.



Δ X>0?
NO
YES
XLOC ≤ LTLIM
XLOC ≥ RTLIM
NO
YES
YES
NO
X = -Δ X
XLOC = XLOC +Δ X
DEFINE X LOCATION OF B-SCAN USING XLOC
STORE DISPLAY INSTRUCTION IN DISPLAY LIST
RETURN

Variables:

RTLIM	=	Right Limit of B-Scan
LTLIM	=	Left Limit of B-Scan
$\Delta X = \pm 5$	=	Movement of B-Scan During One Cycle (1/40 Second)
XLOC	=	X Location of B-Scan

Figure 10. B-Scan Flow Diagram

(The B-scan is simulated by moving it 5 rasters (5/128 inch) each 1/40 second. With this frequency, the eye cannot distinguish discrete movement from continuous motion.)

Student responses also are recorded, and which responses to record constitutes an important decision. An enormous amount of response and response-related data can be recorded by a minicomputer. Since these data must be analyzed by computer programs, and statistical analysis programs tend to be complicated and tedious to write, a problem is to determine the minimum amount of student response data that is necessary to record for analysis.

A program to record responses must code response categories in such a way that they can be kept separate during processing, take care of any temporary buffering of output, and pass the data to the program controlling recording on disk or other medium.

If values of large numbers of response variables are to be recorded, coding to identify variables can become fairly complex, as Appendix A shows. All of the response variables shown there were originally defined from an analysis of the man-machine interface interactions in the RIO's Air Intercept individual skills trainer. There are instructional features unique to the trainer to which the student must respond.

Student Data: Student SHC. Recall the definition of a student sufficient history, given by Atkinson and Paulson (see page 6). There must be a program to compute this sufficient history for whatever purpose it is needed. As we illustrated above, the need may be for input to an optimization program, or it may be to satisfy training managers, or it may be for data for a research design. It is clear that there could be a variety of different sufficient histories, so the problems for generating specifications are to decide what the need is, and then how to compute the sufficient histories that are required.

Computing a student sufficient history for use by an optimization model is a complicated matter, as Atkinson and Paulson (1972) and Chant and Atkinson (1973) have made clear. For an illustration of another approach see Wollmer (1973).

We emphasize that several sufficient histories may be needed for each student. Descriptive analysis of "what went on" during a field trial, for example, may require that values be recorded for a large number of variables, whereas an adaptive controller may need, instead, some summary index. For example, the trials-to-criterion logic we have implemented in the RIO trainer used a "student word" and looked at that word only while the student was engaged by the system (see Figure 11). When the student was transitioned to the next level, the word was cleared, since the need for this information was only temporary.

Student Data: Student Records. As indicated in the CAI system elements diagram (see Figure 2), student records contain those student sufficient histories that are needed for more than temporary purposes. They are stored in a memory until they are needed either by the Adaptive Controller or the Statistical Analyzer. Student records would most likely be stored on disk files. A temporary expedient, which has been used for the RIO trainer, is to punch them on paper tape for later analysis, but this is in the category of "what to do until the disk arrives." The slow punching and

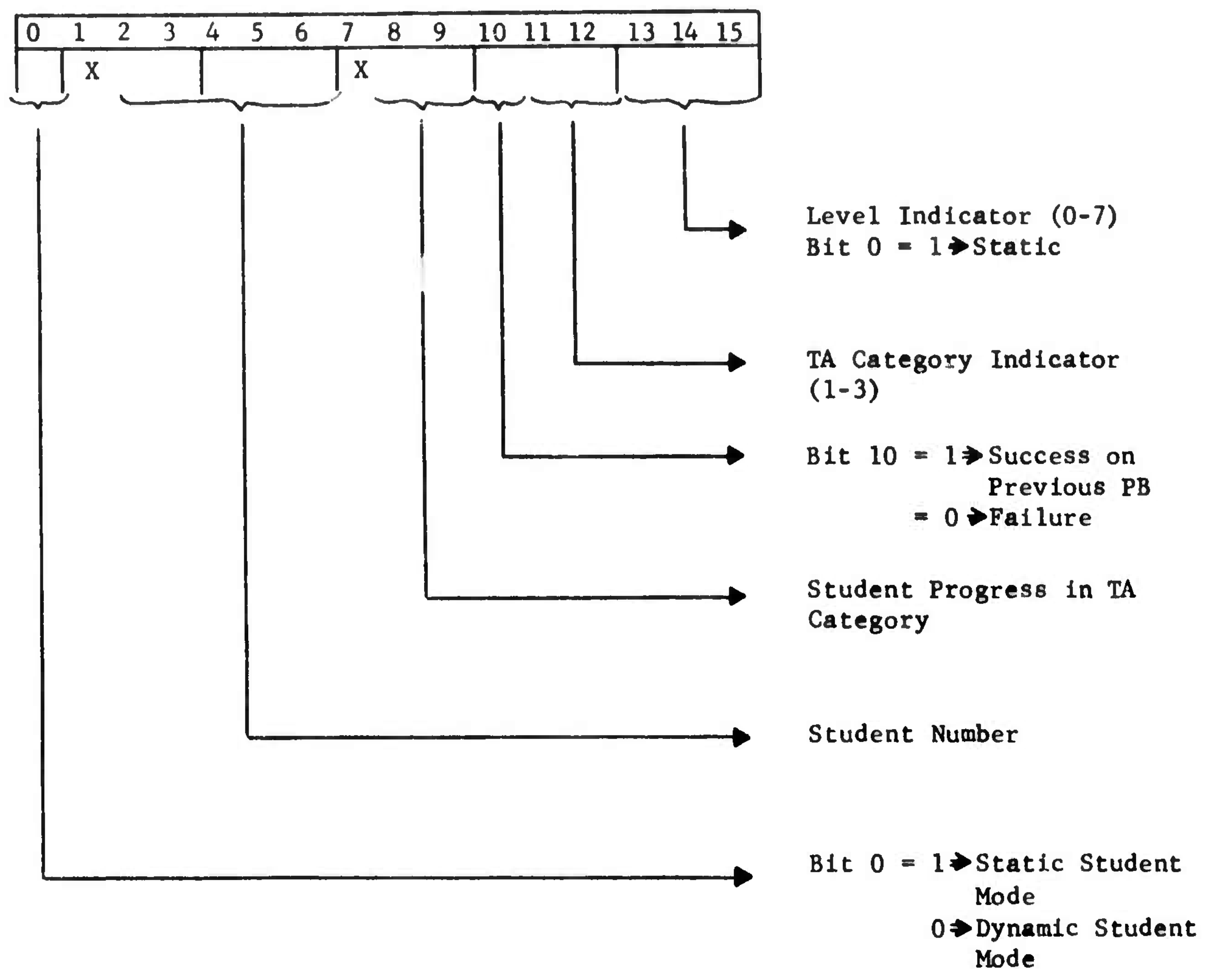


Figure 11. Example of a student sufficient history used for trials-to-criterion logic. Information in the word is compared for a match with criteria for transitioning to the next level

slow reading of paper tape, combined with the possibilities for mislabelling, physical damage, or getting lost in the mail, make this an unreliable and expensive process. Storing student data on a disk file requires a disk-controller and a data-formatter. These operations are performed partly by hardware and partly by software, both general purpose. The programming requirement specific to a particular application is for a routine that will "talk" to the general program and pass data to it in strings that will fit a general format; e.g., a record length, and that will read records from disk to core when they are required.

In the RIO trainer, the data-capture routine sends data to a teletype paper tape. It sends data after the static mode and after the fire display for the dynamic mode. In order to maintain the sequential continuity of the problem, data is first stored in a double-buffer, and then dumped during display cycles when the teletype is free. In this manner, the problem continues running while data is being stored on paper tape.

The appropriate criterion (static or dynamic) subroutine is called by the RIO program; then the data capture subroutine is called from the criterion check subroutine. There are two lists of pointers, one each for the static and dynamic phase data. When called, this subroutine accesses the words indicated by the pointers, and uses a subroutine to store the word in a teletype compatible-format in the teletype buffer. Dumping the buffer occurs automatically; when filled, it will start sending data to the teletype until it is emptied, at which time the dumping routine waits until more data is stored in the buffer.

<u>Adaptive Controller</u>: <u>IS Scheduler</u>. Whether or not the IS Scheduler is a computer program separate from the IS Generator (and IS Optimizer) depends on a number of factors. Scheduling, that is, organizing problems or "frames" into a sequence can be the result of, or the output of these other functions. Whether control over the sequence resides in the student, in the CAI system, or in both influences this. In PSO CAI, a mixed-initiative form of control is used which is truly response-sensitive. That is, the fine-grained instructional sequence is unique to each student. It is convenient for the scheduler to be an executive that controls the generator routines. However, in forms of adaptive control such as dynamic programming, that are basically norm-referenced control, it might be more convenient to combine the scheduler with the optimizer.

It is worth noting that an important part of IS scheduling is keeping track of where the student is. In a course of any length, students will not be able to finish in one session. Any one terminal will be used by a number of different students. There must be logic to identify who is on the terminal, to start the student again at the right place, and to store "where he is" information continuously, so that unscheduled interruptions will not require that the student start all over at the beginning. Some of this information would be passed to the student data programs.

Most PSO-CAI IS Scheduler programs cycle through a "loop" of things to do, since PSO CAI is concerned with giving students practice in performing certain well-defined tasks or problems. The IS Scheduler under these circumstances is the top-level executive program. An example of an IS Scheduler,

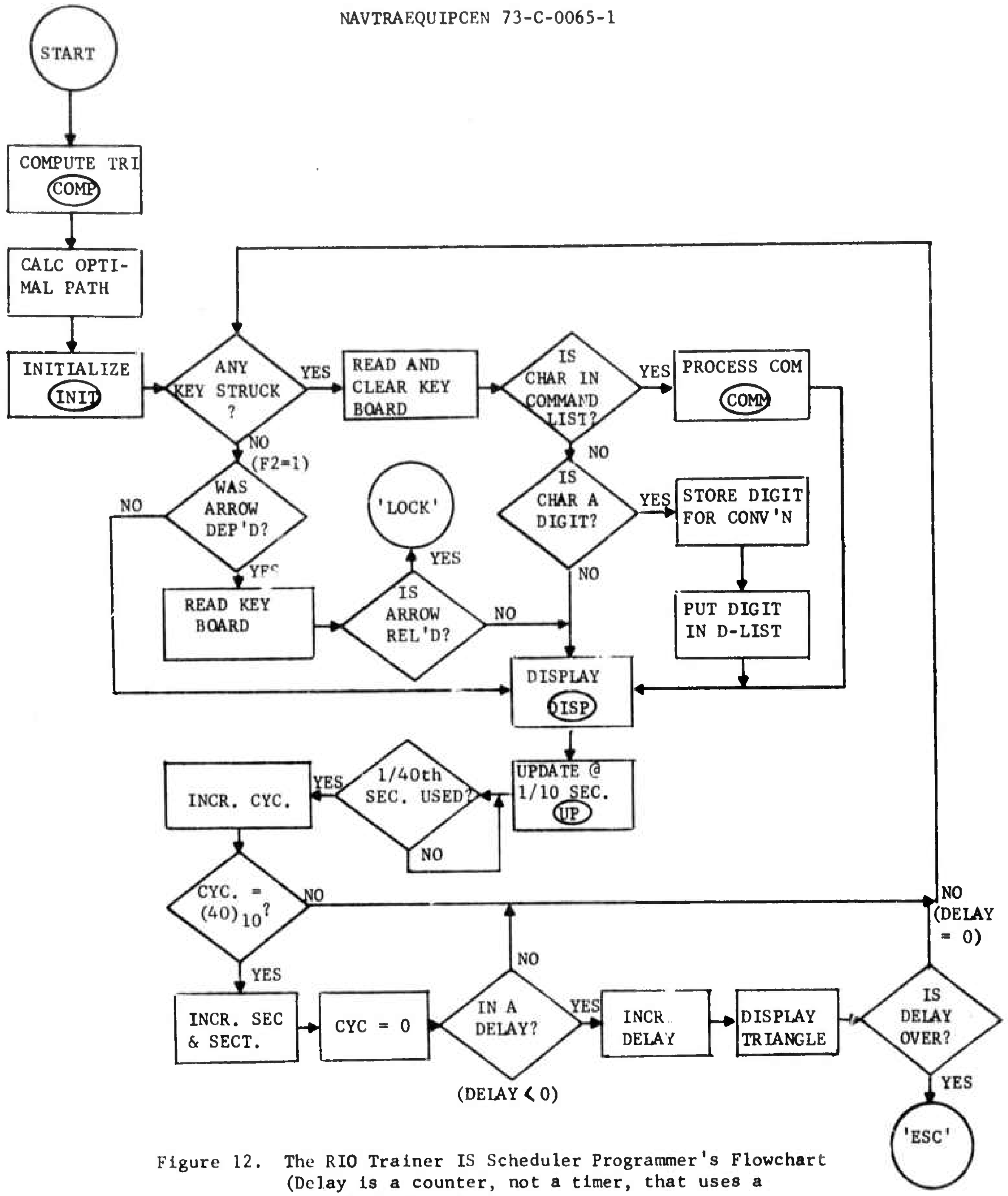


Figure 12. The RIO Trainer IS Scheduler Programmer's Flowchart (Delay is a counter, not a timer, that uses a special instruction)

for the RIO trainer, illustrated by the following programmer's flowchart (Figure 12).

Adaptive Controller: IS Generator. We assume that the IS Generator is a program that creates the "within problem" interaction with the student while he is performing, and that the instructional sequence has not be predetermined in the form of programmed instruction by an army of course authors. This generator contains the logic for interacting with the student, which must necessarily include calling subroutines or other programs that generate stimulus displays or record responses.

How general can the IS Generator be? We have developed and demonstrated IS Generators for a class of problem-solving activities called "troubleshooting" (Rigney, et al., 1972a). Limits of this class have not been explored, since there has only been time to develop data-structures for electronic and electromechanical devices. We see no particular reason why the class could not include medical "troubleshooting;" i.e., diagnostic activities for animate as well as for inanimate devices. We do know these IS Generators, in their present form, are restricted to tasks in which the student must perform information-sampling operations on some malfunctioning system, interpret the information from these tests, and revise his hypothesis set about possible causes of the malfunction until, by successive approximations, he finds the actual cause.

We also have developed and demonstrated an IS Generator that is general to a class of procedural tasks which are concerned with changing the states of devices by manipulating inputs to the devices. Again, the boundaries of the class have not been explored. The general requirement is that the structure of the task the student is to learn be representable in the form of a final goal, and intermediate goals, linked to each other and to actions to be performed to accomplish goals, by any of a group of logical operators which define the task-structural relationships (Rigney, et al., 1972a).

These programs have taught us certain facts about generality requirements. First, there must be some general task-structure that is to be the basis for interaction between the program and the student. A general structure can be defined for troubleshooting, and for procedural tasks. Second, there must be some general way of representing states of devices being "worked-on," of student progress, of problems and procedures -- so that these states can be continuously updated as they change. This requires also that it be possible to represent relationships among events, conditions affecting states, and states.

Third, it must be possible to describe essential features of specific devices and specific tasks in terms of general data-structure formats and codes. The program then can operate on these data structures in the same ways, regardless of the particular device or tasks involved, until it communicates with the external world; i.e., the student, when it then takes the additional step of displaying a specific lexical word or phrase that is matched to an internal code.

Fourth, the student must inform the program of what he is trying to do. This amounts to the student telling the program what his next goal is going to be. In troubleshooting, this would be the next problem he is going to take, or the next indicator he is going to sample for symptom information. Knowing what goal the student is working toward allows the program to track his progress toward that goal, offer assistance, and provide progress information.

It is important to appreciate the complexity of the student-program interaction that is required for PSO CAI. This is no multiple-choice question game. The programmer's flowchart for two of the programs in the IS Generator for the RIO trainer are illustrated in Figure 13.

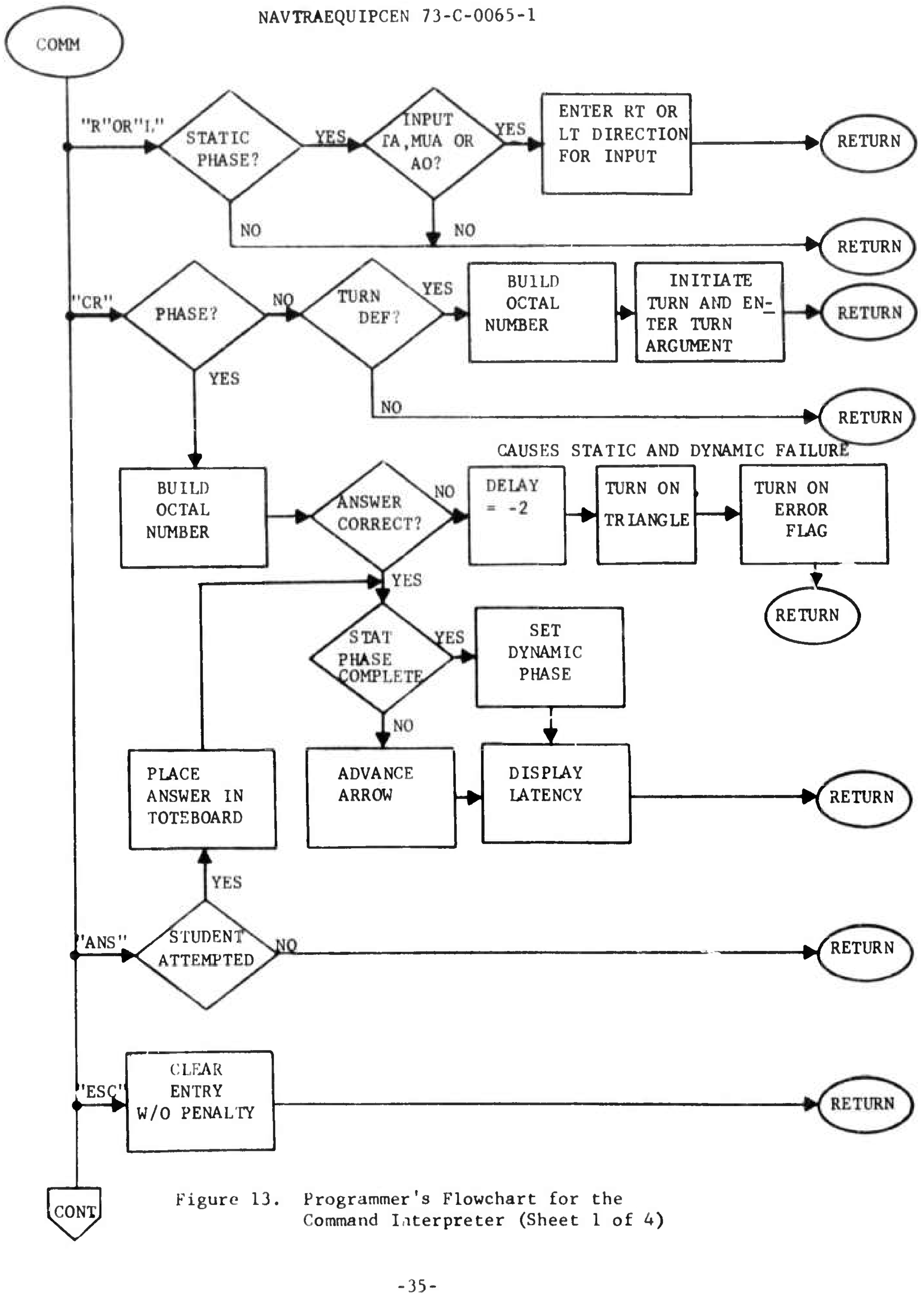


Figure 13. Programmer's Flowchart for the Command Interpreter (Sheet 1 of 4)

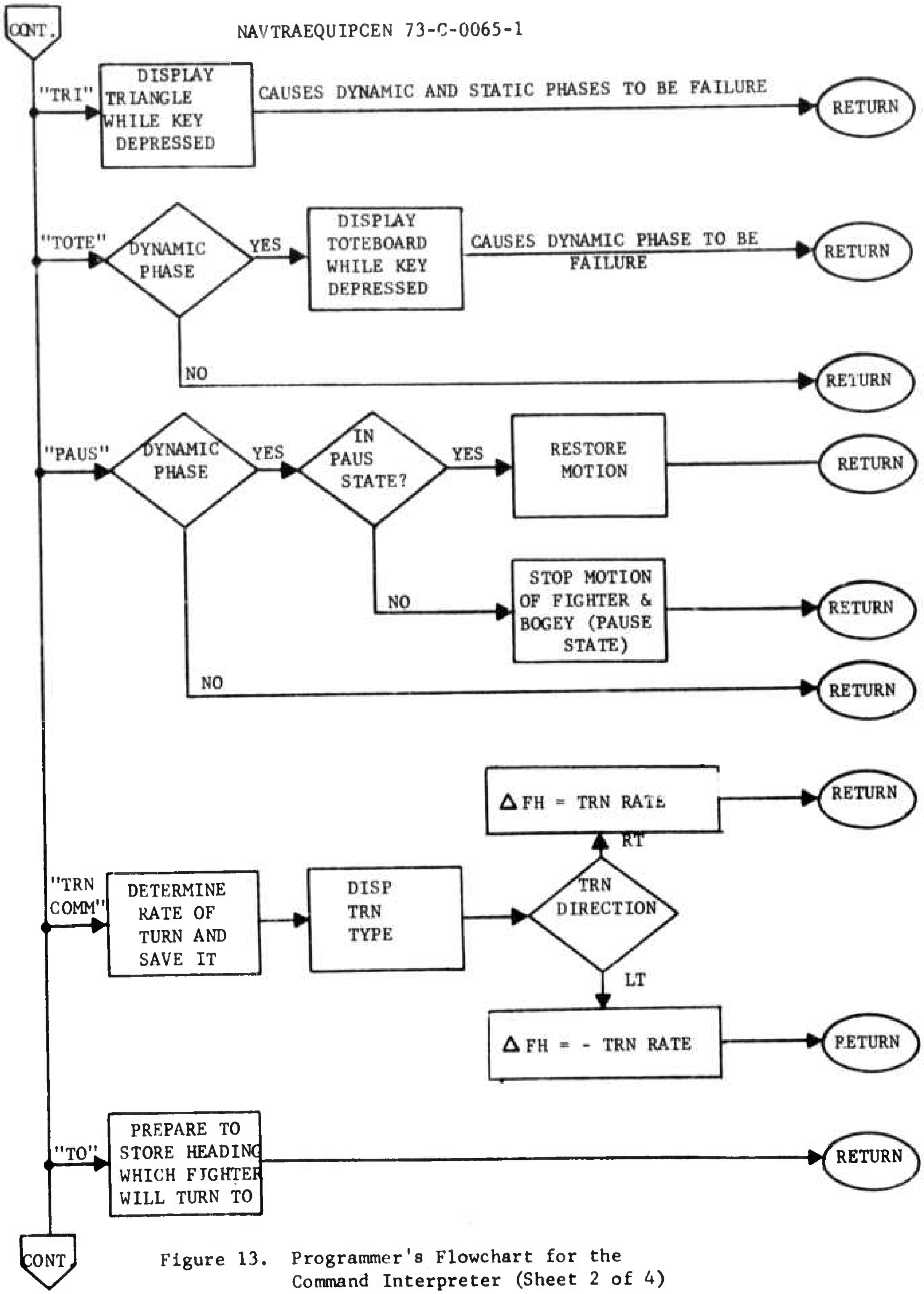

CONT.
"TRI"
DISPLAY TRIANGLE WHILE KEY DEPRESSED
CAUSES DYNAMIC AND STATIC PHASES TO BE FAILURE
RETURN
"TOTE"
DYNAMIC PHASE
YES
DISPLAY TOTEBOARD WHILE KEY DEPRESSED
CAUSES DYNAMIC PHASE TO BE FAILURE
RETURN
NO
RETURN
"PAUS"
DYNAMIC PHASE
YES
IN PAUS STATE?
YES
RESTORE MOTION
RETURN
NO
STOP MOTION OF FIGHTER & BOGEY (PAUSE STATE)
RETURN
NO
RETURN
ΔFH = TRN RATE
RETURN
RT
"TRN COMM"
DETERMINE RATE OF TURN AND SAVE IT
DISP TRN TYPE
TRN DIRECTION
LT
ΔFH = - TRN RATE
RETURN
"TO"
PREPARE TO STORE HEADING WHICH FIGHTER WILL TURN TO
RETURN
CONT

Figure 13. Programmer's Flowchart for the Command Interpreter (Sheet 2 of 4)

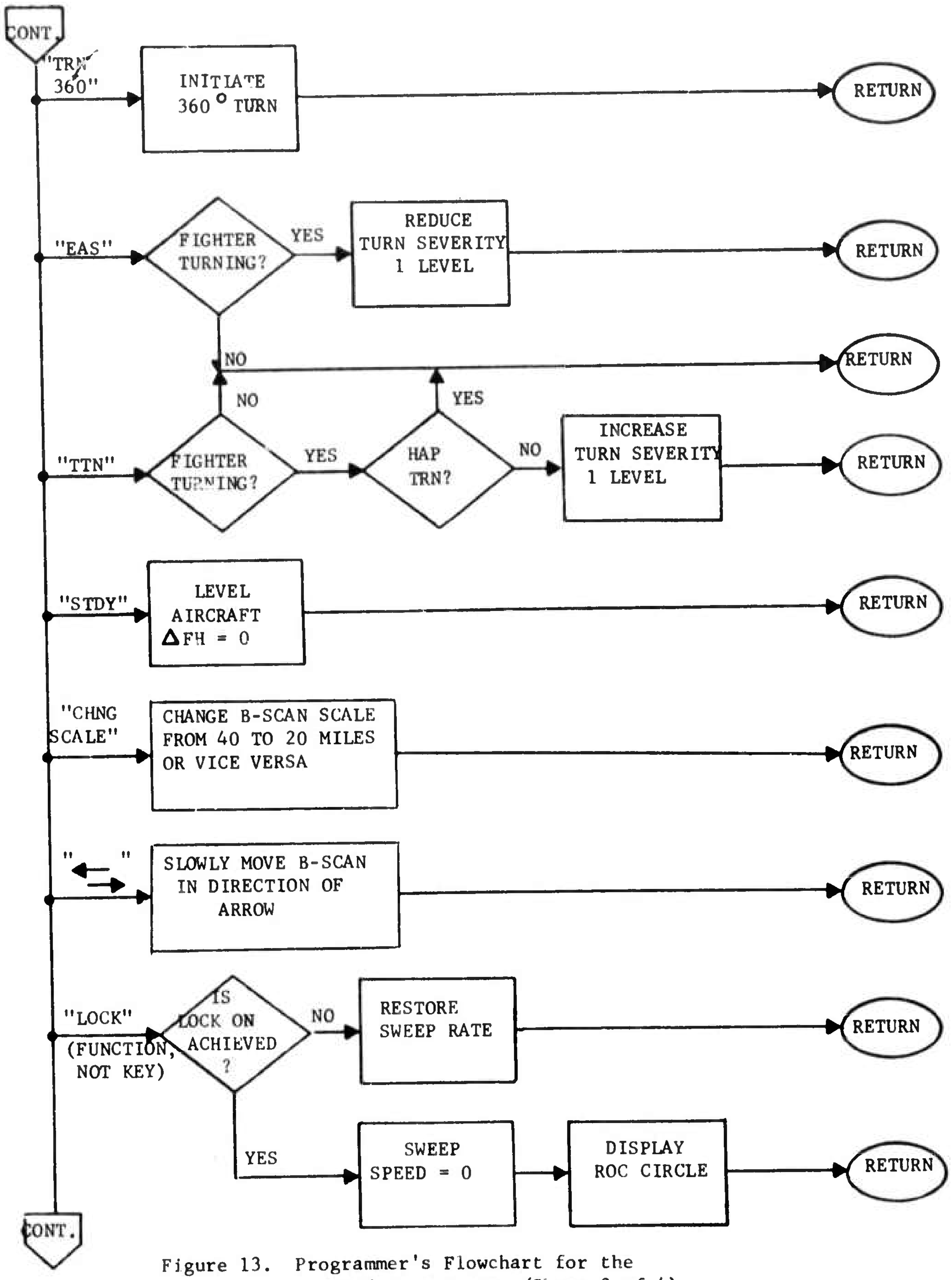


Figure 13. Programmer's Flowchart for the Command Interpreter (Sheet 3 of 4)

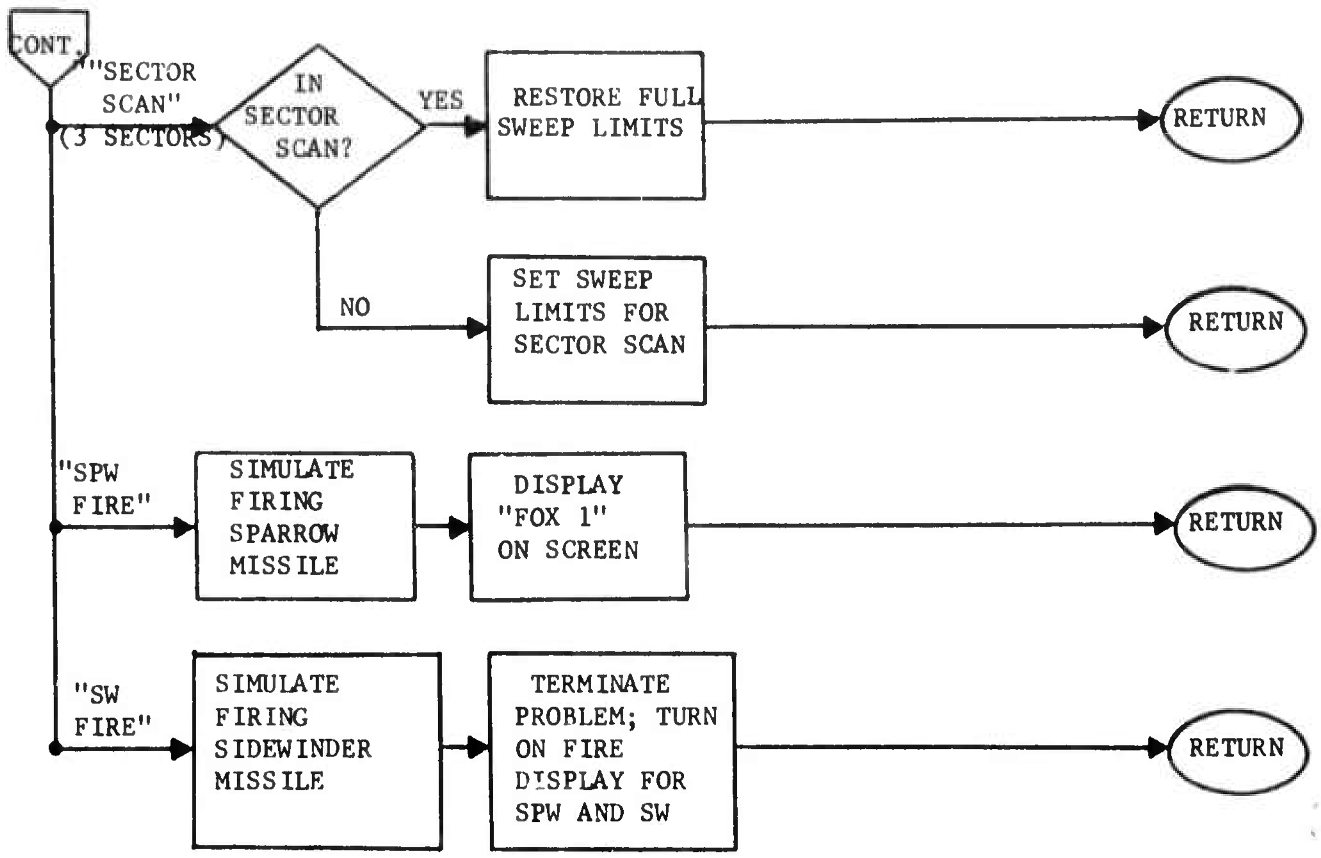

CONT.
"SECTOR SCAN" (3 SECTORS)
IN SECTOR SCAN?
YES
RESTORE FULL SWEEP LIMITS
RETURN
NO
SET SWEEP LIMITS FOR SECTOR SCAN
RETURN
"SPW FIRE"
SIMULATE FIRING SPARROW MISSILE
DISPLAY "FOX 1" ON SCREEN
RETURN
"SW FIRE"
SIMULATE FIRING SIDEWINDER MISSILE
TERMINATE PROBLEM; TURN ON FIRE DISPLAY FOR SPW AND SW
RETURN

Figure 13. Programmer's Flowchart for the Command Interpreter (Sheet 4 of 4)

Legend:

The COMM program is "called" by the IS Scheduler illustrated in Figure 12. The COMM program accepts commands and other inputs from the keyboard. These student inputs are described as follows:

DIGITS	-	Numerical values are entered for the toteboard (static) phase, and for the definition of turns in the dynamic phase. If a number is entered immediately after the type and severity of a turn is defined, it represents the number of degrees to turn. If the "TO" key is used, then the number represents the heading at which the fighter should complete the turn.
R and L	-	These keys are used to define right or left for the input of TA, MUA and AO.
CR	-	Carriage return is used to define a completed entry either for the toteboard or the definition of a turn.
ANS	-	The answer key is used only during the static phase (filling in the toteboard). If the student cannot solve the problem for a given entry, then the answer key will provide the correct answer (the student must have tried at least one entry).
ESC	-	The escape key is used to erase an incorrect entry. It does not penalize the student if he makes a keyboard error and then uses the escape key before hitting carriage return. For a turn entry, the escape key will clear the turn display and level the aircraft.
TRI	-	The triangle displays the geometric "birds eye" view of the problem on the top half of the CRT. It includes the optimal path and a trace of the bogey's path on the B-scan display.
TOT	-	The toteboard displays the current toteboard values. It can be used only during the dynamic phase of the problem.
PAUSE	-	The pause key stops the motion of the fighter and bogey, thus "freeing" the problem until the pause key is depressed again. The key can be used only during the dynamic phase of the problem.
TURN COMMAND	-	There are eight turn commands, four each for port and starboard. They specify four levels of severity (easy, standard, hard, and hard-as-possible). The turn command also puts the program in a mode to receive the numerical input specifying the parameters of the turn.
TO	-	The TO command is used immediately following a turn command. It is used to define the heading to which the RIO wants

the aircraft to turn. If this command is not used, the number input will be interpreted as the number of degrees the aircraft is to turn before leveling.

TRN 360	-	The TURN 360 key is used to turn the aircraft 360°, leaving the aircraft at its present heading after the turn. Typically, this turn is terminated by a STDY command.
EAS	-	The EASE key reduces the severity of a turn one level (e.g., a standard turn reduces to an easy turn, an easy turn reduces to level flight). If the aircraft is not in a turn, the EASE key does nothing.
TTN	-	The TIGHTEN key increases the severity of the aircraft turn, unless the aircraft is either in a hard-as-possible turn or is not turning. In this case this key does nothing.
STDY	-	The STEADY command stops a turn and levels the aircraft.
CHNG SCALE	-	The CHANGE SCALE affects the B-scan range. The range initially starts at 40 miles range, but it can be switched to 20 miles. This key changes the range of the radar from 40 to 20 or from 20 to 40 when it is depressed.
$\rightleftarrows$	-	These two keys simulate the ability to control the radar scan manually. The scan is in the direction of the arrow at a slow rate.
LOCK	-	The LOCK function occurs when the bogey is on the B-scan display and one of the two $\rightleftarrows$ keys was just released within 3° of the bogey. This displays the ROC (rate of closure) circle and keeps illuminating the target until it leaves the range and angle off limits of the B-scan.
SECTOR SCAN	-	There are three areas of sector scan: Left (45° left - 8° right); Center (23° left - 23° right); Right (8° left - 45° right). When the scan is so restricted, it naturally paints the bogey more frequently.
SPW FIRE	-	The SPARROW FIRE key simulates the firing of a Sparrow Missile. The display list for the fire display is defined by the computer and stored until the sidewinder is fired. The indication FOX 1 is displayed on the screen to signify that the Sparrow has been fired.
SW FIRE	-	The SIDEWINDER FIRE key simulates the firing of a Sidewinder Missile, terminates the problem, and presents a fire display for the Sparrow (if fired), and Sidewinder Missile. The criteria for success or failure is evaluated depending upon speed level and the hit probabilities of the two missile firings. When the fire key is depressed again, the computer selects the next appropriate problem for the student.

FREE FLY	-	The FREE FLY key is available only when the fire display is present. It gives the students the problem previously completed in the free fly mode.
STOP	-	The STOP key is used when the student has completed a session; the teletype buffer is dumped on paper tape, and leader is punched allowing the operator sufficient room to appropriately label and tear off the tape.

The COMP program (see Figures 12 and 14) does the computations necessary to "run" the intercept problems. Basically, this is a matter of updating the positions of the bogey and fighter on the CRT. The values of variables that are computed by COMP are used in the computer graphics programs to generate the real-time displays, and for tracking the student's progress, so that his values of intercept triangle variables and his firing positions can be scored right or wrong.

The subroutine COMP computes the angular relationships, distance relationships, and calls the subroutine to compute the rate of closure for the fighter and the bogey. This requires the X and Y position of the two aircraft, and both headings. The computation routine is strictly a static routine, it calculates X and Y, which provides sufficient information to determine the bogey bearing with the use of a table of tangents. Given bogey bearing and the aircraft headings, all angular relationships can be calculated. Range and displacement distance is calculated using X, Y, computed angular relationships, and a sine-cosine table.

Motion is simulated by moving the X and Y locations of the fighter and bogey as appropriate for their speed and direction. Their positions are updated every 1/10 second, at which time the COMP subroutine is run to keep all angular and distance values current. It has been determined that greater frequency of updating would not noticeably improve the fidelity of the simulation.

Adaptive Controller: IS Optimizer. Aside from Atkinson's group we know of no day-to-day applications of quantitative process-control models in CAI. We are using relatively crude, multi-stage trials-to-criterion logic for adapting to individual differences. This has been demonstrated to be effective, in the sense that more able students require less practice to finish the course than less able students, and the group variances are reduced to a very narrow range at the end of the course (Rigney, et al., 1973b).

The logic for doing this is a part of the IS Generator-Scheduler programs. Since it is relatively simple, there is no particular reason to separate it. Criteria for latency, errors, and accuracy can be set. Student words are updated with values of these variables as the student works. The IS Optimizer logic looks at this student sufficient history after each problem (for example, see Figure 9). When the values in the words match the criteria, the student is transitioned to the next level.

A Trial-to-Criterion Strategy for the RIO Trainer. This strategy requires the student to continue attempting practice problems until he has satisfied the criterion to advance to the next level, repeating this process in new categories until the course is completed. The basic criterion is to complete two consecutive problems in each target aspect category (TAC) successfully without the use of any of the instructional aids available on key demand.

The levels are structured as follows:

1. The first level is an introductory level where the student is given one free fly problem at each of 3 target aspect categories.

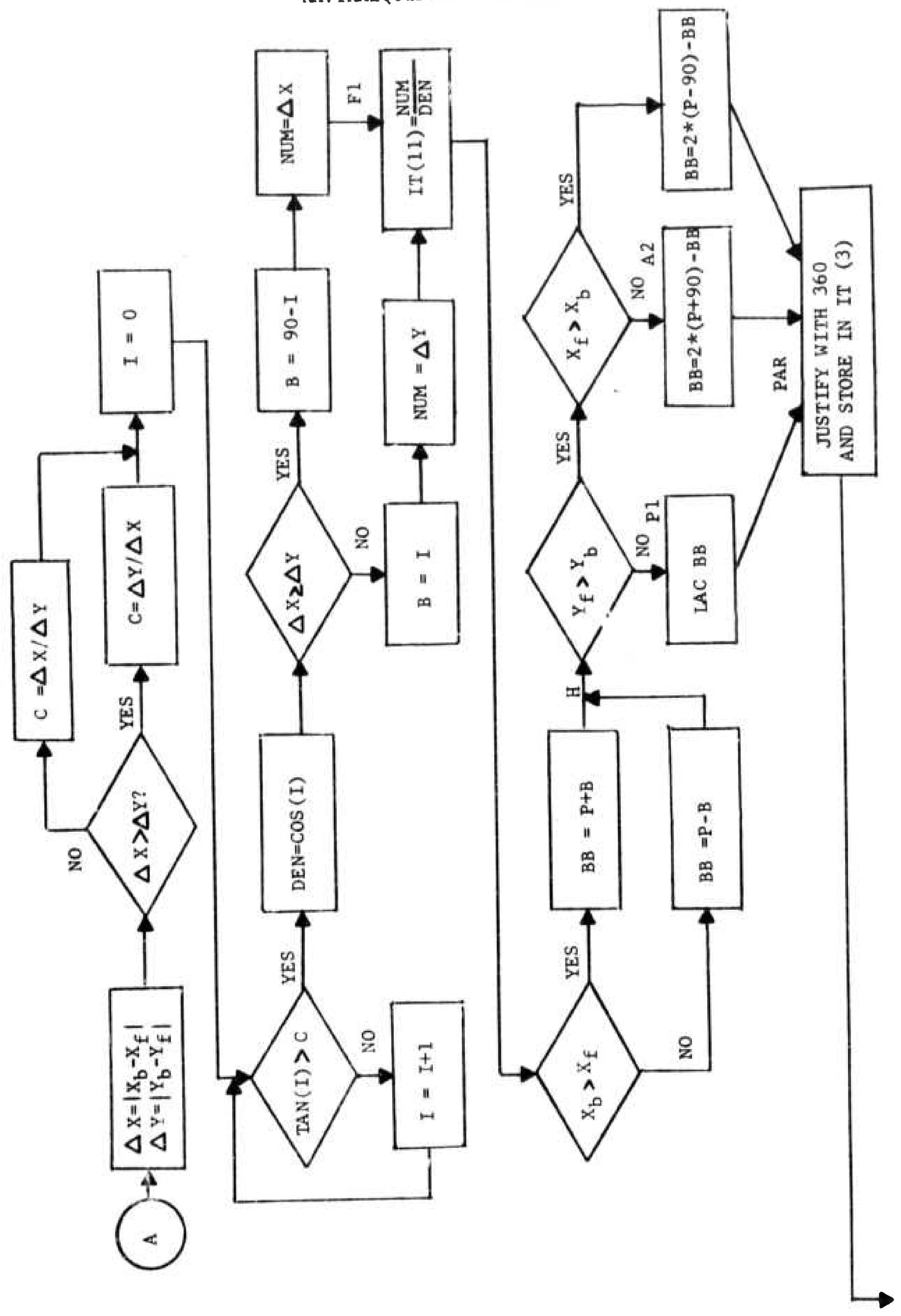


Figure 14. Programmer's Flowchart for the COMP Subroutine (Sheet 1 of 2)

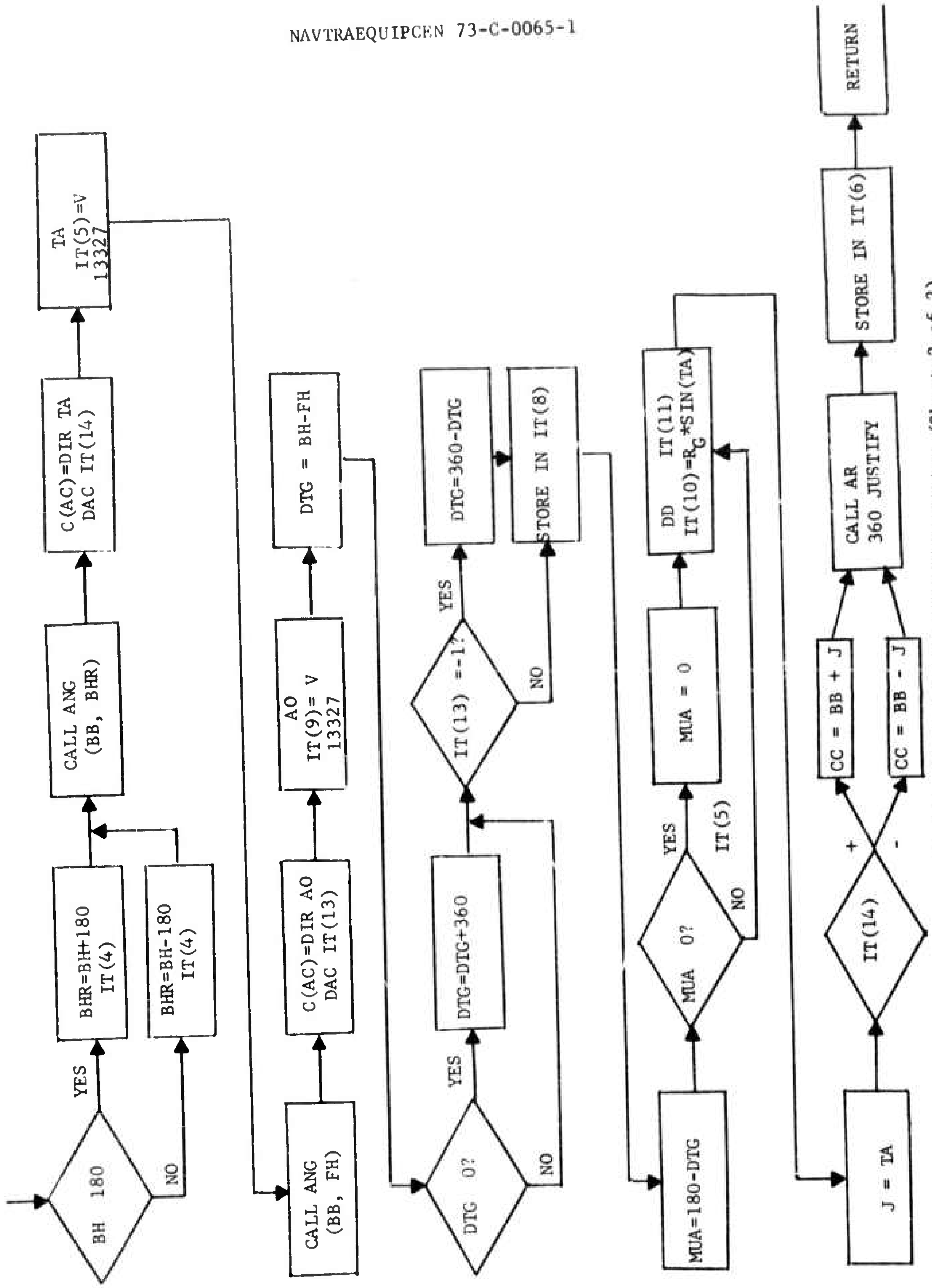


Figure 14. Programmer's Flowchart for the COMP Subroutine (Sheet 2 of 2)

2. The next two levels are static problem levels. The first static level displays the triangle while the student fills in the toteboard. In the second static level the triangle is eliminated. In each level, the student must complete two consecutive problems without error in 60 seconds or less for each of 3 target aspect categories.

3. The next two levels are dynamic problems at a speed of 220 and 260 knots, respectively. The criteria for these two levels are to achieve a hit probability of .80 or better for the Sidewinder fire; a Sparrow fire is not required. The toteboard must be completed without error. Each of three target aspect categories begins with a free fly problem.

4. The next five levels are dynamic problems at 300, 340, 380, 420, and 460 knots, respectively. The criteria for success include that of 220 and 260 speed levels and also require a Sparrow hit probability of .50 or better. Each level has 3 target aspect categories.

5. The last level is the transfer test. It includes 20 problems which are run at 500 knots. The target aspects of these problems include low target aspect problems and problems with target aspects not used for the three target aspect categories for practice problems. The student has no aids available, and advances to the next problem regardless of the results of previous problems. The conclusion of these problems is the end of this course.

Content/Mediator Files. These are the data-structures the IS Generator-Scheduler operates upon to produce the instructional sequence. For PSO CAT, they tend to be fairly simple lists of specifications describing selected features of devices, tasks, or problems. Although the formats of these data-structures can be general, as indeed they must be for the IS Generators discussed above, their contents obviously must be specific to the application.

As we said before, it is feasible to produce interpretive programs to allow non-programmer personnel to prepare data-structures. Otherwise, a team of a subject-matter expert and a computer programmer must be formed. Since the subject-matter expert is likely to view the world in an entirely different way than the computer programmer, a certain amount of discussion will be required to extract the needed information.

In the case of the RIO trainer, many practice intercept problems were generated from a basic problem, using an instruction to the program having the following format:

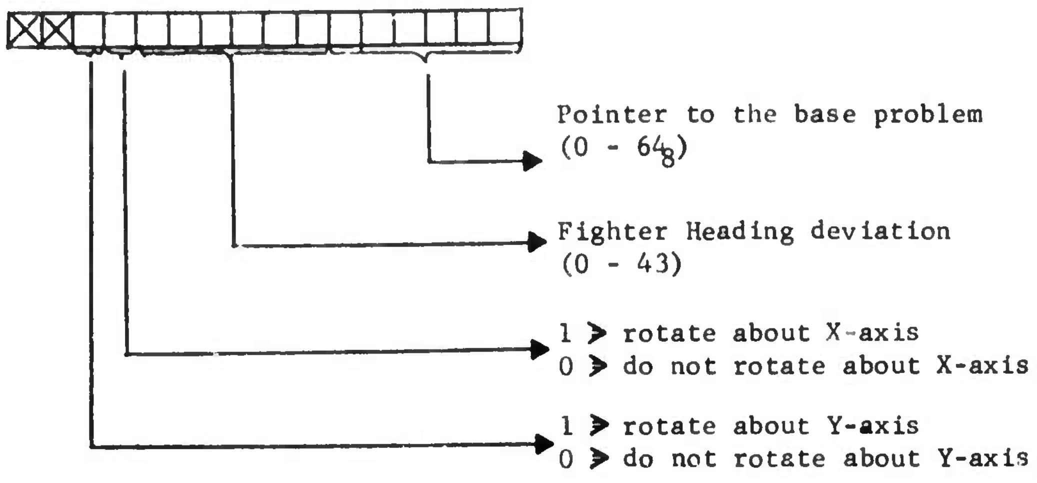

Pointer to the base problem
(0 - 64_8)
Fighter Heading deviation
(0 - 43)
1 > rotate about X-axis
0 > do not rotate about X-axis
1 > rotate about Y-axis
0 > do not rotate about Y-axis

Figure 15. The problem pointer word used in generating practice problems

There are 11 basic problems for each target aspect category. Each problem may be rotated about the X-axis, Y-axis, or both. These base problems define the initial X and Y positions of the fighter and bogey and their headings. The fighter is set on a collision course for the basic problems.

The problem pointers specify the fighter heading change, the base problem, and the rotation. The deviation from collision course is a variable that affects the difficulty of the B-scan interpretation after the toteboard is completed. A deviation of 90° or 270° starts the fighter perpendicular to the collision course, causing the target aspect to change rapidly while the student is entering toteboard values and subsequently waiting for the fighter to reach collision course. This variable is restricted for the first few speed levels, then varied more freely after the student has had a chance to familiarize himself with the procedures and the trainer. Problems at the start of a target aspect level are given a higher deviation, as they are flown in free-fly and hopefully serve to illustrate the effects of drift while achieving collision course.

There is a total of 128 practice problems for each target aspect, or a total of 384 problems for the 2 static and 7 dynamic levels. The 20 test problems have 20 base problems and 20 pointers. This is necessary as the characteristics of the transfer test problems are more vigorously controlled than the practice problems. Also for the test problems, it was deemed more important to consider opening or closing target aspect, and crossing the bogey flight path for low target aspect problems.

The purpose of computer-controlled problem selection is not only convenience of operation, but is primarily to reduce uncontrolled variation for the statistical analysis of the problem. When each student starts a given target aspect category within a speed level, he will start with the same problem as every other student, in both "stripped" and "enhanced" versions of the trainer. Obviously, due to advancement based on a trials-to-criterion strategy, each student will require a different number of problems to satisfy the criterion.

Instructional Sequence. Since in PSO CAI this is "made up" by the Adaptive Controller operators, it literally does not exist until the moment of use. Its major constituents, content, knowledge of results, content mediators, and L & M mediators, are produced by computer programs operating on data-structures and student inputs.

Statistical Analysis. These programs could be standardized to accept sufficient histories from the SHC programs. Once students-by-variables arrays are available, standard statistical analysis programs could be used. Sums, sums of squares, and N's for individual students and for samples and subsamples are useful intermediate products to have a data-summarizing program printout. These allow convenient visual inspection of the data. This may reveal information or suggest analyses that may not have been foreseen.

Where multiple dependent variables and multi-mode, multi-difficulty level instructional structures are used, programs to analyze data become relatively complicated. Computations are simple, but data must be partitioned into many different categories. The short word length of minicomputers; e.g. sixteen bits, is not convenient for accumulating sums and sums of squares, or for doing multiply-divide. Floating point or multiple-precision logic is required. The poi[illegible] are making is that these data-processing programs require a substantia[illegible]ortion of programmer time to produce.

SECTION IV

SUMMARY AND CONCLUSIONS

This report contains a description of PSO CAI, and general procedures to follow in implementing PSO CAI. Key personnel are subject-matter experts, instructional technologists, and computer programmer/analysts.

The six elements required in any CAI system were described. Each of these must be represented in the design of a CAI system. Each, except the student element, contains operations that must be implemented either by hardware or by software. Some illustrations of how these operations were implemented in the RIO trainer are given, in the form of examples of task analyses, instructional flowcharts, and of the computer programmer's flowcharts that were derived from them.

We considered how a "general operating system" of computer programs, and a general hardware system, for all applications of PSO CAI might be produced. The most obvious general approaches to software production currently visible are the development of interpreters and compilers, which would permit more rapid production of computer graphics, computer programs, and data-structures. For teaching well-defined classes of job skills, Instructional Sequence Generators, which constitute roughly fifty to ninety percent, depending on the amount of computer graphics, of the computer programs needed, can be general within a class of job skills. The TASKTEACH Instructional Sequence Generator is an example (Rigney, et al., 1972a). Conditions necessary for this kind of IS Generator are:

1. There must be some general way for describing task-structures that can be the basis for interaction between the program and the student. General structures have been defined for troubleshooting, and for procedural tasks.

2. There must be some general way of representing states of devices being "worked-on," of student progress, of problems and of procedures -- so that these states can be continuously updated. This requires also that it be possible to represent relationships among events affecting states, and different states.

3. It must be possible to describe essential features of specific devices and specific tasks in terms of general data-structure formats and codes. The program then can operate on these data-structures in the same ways, regardless of the particular devices or tasks involved, until it communicates with the external world; i.e., the student, when it then takes the additional step of displaying a special lexical word or phrase that is matched to a general internal code.

4. The student must inform the program of what he is trying to do. This amounts to the student telling the program what his next goal is going to be. In troubleshooting, this would be the next problem

he is going to take, or within a program, the next indicator he is going to sample for symptom information, or the next front-panel test drill he is going to take. Knowing what goal the student is working toward allows the program to track his progress toward that goal, offer assistance, and provide progress information.

A general hardware system for supporting PSO CAI was described. With the exception of some IO interfacing, its components are off-the-shelf commercial items.

REFERENCES

Anderson, R. C. Encoding processes in the storage and retrieval of sentences. Journal of Experimental Psychology, 1971, 91, 338-340.

Anderson, R. C. Concretization and sentence learning. Unpublished manuscript, University of Illinois, 1973.

Anderson, R. C., & Hidde, J. L. Imagery and sentence learning. Journal of Educational Psychology, 1971, 62, 526-530.

Anderson, J. R., & Bower, G. H. (Eds.) Human Associative Memory. Washington, D. C.: V. H. Winston and Sons, 1973.

Atkinsin, R. C., & Paulson, J. A. An approach to the psychology of instruction. Psych. Bull., 1972, 78(1), 49-61.

Begg, I., & Paivio A. Concreteness in sentence meaning. Journal of Verbal Learning and Verbal Behavior, 1969, 8, 821-827.

Bloom, B. S. (Ed.) Taxonomy of Educational Objectives: I. Cognitive Domain. New York: David McKay, 1956.

Bower, G. H. Mental imagery and associative learning. In Lee Gregg (Ed.), Cognition in Learning and Memory. New York: Wiley and Sons, 1972.

Bryan, G. L., Rigney, J. W., Orr, W. K., & Svenson, D. W. An analysis of destroyer CIC officer duties in anti-submarine operations. Los Angeles: Electronics Personnel Research Group, February 1956a.

Bryan, G. L., Rigney, J. W., & Svenson, D. W. An analysis of CIC officer duties in air intercepts, radar, navigation, and shore bombardment. Los Angeles: Electronics Personnel Research Group, March 1956b.

Bugelski, B. R. Images and mediators in one-trial paired-associate learning. II: Self-timing in successive lists. Journal of Experimental Psychology, 1968, 77. 328-334.

Bugelski, B. R., Kidd, E., & Segmen, J. Image as a mediator in one-trial paired-associate learning. Journal of Experimental Psychology, 1968, 76, 69-73.

Carbonell, J. Scholar, A New Approach to Computer-Assisted Instruction. Naval Research Review, October 1972.

Chant, V. G., and Atkinson, R. C. Optimal allocation of instructional effort to interrelated learning strands. Journal of Mathematical Psychology, 1973, 10, 1-25.

Chomsky, N. Current issues in linguistic theory. In J. A. Fodor and J. J. Katz (Eds.), _The Structure of Language_: _Readings in the Philosophy of Language_. Englewood Cliff, N. J.: Prentice Hall, 1964.

Craig, E. M. Role of mental imagery in free recall of deaf, blind, and normal subjects. _Journal of Experimental Psychology_, 1973, _97_, 249-253.

Davidson, R. E. Mediation and ability in paired-associate learning. _Journal of Educational Psychology_, 1964, _55_, 352-356.

Ernest, C. H., & Paivio, A. Imagery ability in paired-associate and incidental learning. _Psychonomic Science_, 1969, _15_, 181-182.

Ernest, C. H., & Paivio, A. Imagery and verbal associative latencies: A function of imagery ability. _Canadian Journal of Psychology_, 1971, _25_, 83-90.

Gagne, R. M. _The Conditions of Learning._ New York: Holt, Rinehart, and Winston, 1970.

Griffith, D., & Johnson, W. A. An information-processing analysis of visual imagery. _Journal of Experimental Psychology_, 1973, _100_, 141-146.

Levin, J. R. Inducing comprehension in poor readers: A test of a recent mode. _Journal of Educational Psychology_, 1973, _65_, 19-24.

Lippman, M. Z., & Shanahan, M. W. Pictorial facilitation of paired-associate learning: Implications for vocabulary training. _Journal of Educational Psychology_, 1973, _64_, 216-222.

Marks, D. F. Individual differences in the vividness of visual imagery and their effect on function. In P. W. Sheehan (Ed.), _The Function and Nature of Imagery_. New York: Academic Press, 1972.

Montague, W. E., & Carter, J. F. Vividness of imagery in recalling connected discourse. _Journal of Educational Psychology_, 1973, _64_, 72-75.

Nelson, D. L., & Brooks, D. H. Functional independence of pictures and their verbal memory codes. _Journal of Educational Psychology_, 1973, _98_, 44-48.

Paivio, A. Mental imagery and associative learning and memory. _Psychological Review_, 1969, _76_, 241-263.

Paivio, A., & Csapo, K. Concrete-image and verbal memory codes. _Journal of Experimental Psychology_, 1969, _80_, 279-285.

Paivio, A., & Madigan, S. A. Imagery and associative value in paired-associate learning. _Journal of Experimental Psychology_, 1968, _76_, 35-39.

Paivio, A., & Madigan, S. A. Noun imagery and frequency in paired-associate and free recall learning. _Canadian Journal of Psychology_, 1970, _24_, 353-361.

Paivio, A., & Okovita, H. W. Word imagery modalities and associative learning in blind and sighted subject. Journal of Verbal Learning and Verbal Behavior, 1971, 10, 506-510.

Paivio, A., & Rowe, E. J. Noun imagery, frequency, and meaningfulness in verbal discrimination. Journal of Experimental Psychology, 1970, 85, 264-269.

Paivio, A., & Smythe, P. C. Word imagery, frequency, and meaningfulness in short-term memory. Psychonomic Science, 1971, 22, 333-335.

Paivio, A., & Yuille, J. C. Mediation instructions and word attributes in paired-associate learning. Psychonomic Science, 1967, 8, 65-66.

Paivio, A., & Yuille, J. C. Changes in associative strategies and paired-associate learning over trials as a function of word imagery and type of learning set. Journal of Experimental Psychology, 1969, 79, 458-463.

Paivio, A., Yuille, J. C., & Rogers, T. B. Noun imagery and meaningfulness in free and serial recall. Journal of Experimental Psychology, 1969, 79, 509-514.

Powers, W. T. Behavior. The Control of Perception. Chicago: Aldine, 1973.

Prytulak, L. S. Natural language mediation. Cognitive Psychology, January 1971, 2(1), 1-56.

Reese, H. W. Imagery in paired-associate learning in children. Journal of Experimental Child Psychology, 1965, 2, 290-296.

Rohwer, W. D., Jr., Lynch, S., Levin, J. R., & Suzuki, N. Pictorial and verbal factors in the efficiency learning of paired-associates. Journal of Educational Psychology, 1967, 58, 278-284.

Rigney, J. W., & Bond, N. A., Jr. Maintainability prediction: Methods and results. Los Angeles: Electronics Personnel Research Group, June 1964.

Rigney, J. W., & Towne, D. M. Computer techniques for analyzing the micro-structure of serial-action work in industry. Human Factors, 1969, 11(2), 113-122.

Rigney, J. W. Implications of research on internal processing operations in learning and memory for serial task training. Los Angeles: Behavioral Technology Laboratories, January 1971.

Rigney, J. W., Towne, D. M., King, Carole A., & Langston, E. T. Computer-aided performance training for diagnostic and procedural tasks. Los Angeles: Behavioral Technology Laboratories, October 1972a.

Rigney, J. W., & Williams, L. A. A general-purpose ASCII decoder for control of peripheral devices for CAI terminals. Los Angeles: Behavioral Technology Laboratories, January 1972b.

Rigney, J. W. A plan for applying performance-oriented CAI in Naval technical training. Los Angeles: Behavioral Technology Laboratories, April 1973a.

Rigney, J. W., Morrison, D. K., Williams, L. A., & Towne, D. M. Description and initial evaluation of a computer-based individual trainer for the radar intercept observer. Los Angeles: Behavioral Technology Laboratories, April 1973b.

Rigney, J. W. A discussion of Behavioral Technology Laboratories CAI projects in relation to a CAI test-bed concept. Los Angeles: Behavioral Technology Laboratories, July 1973c.

Rigney, J. W., & Towne, D. M. An interpreter for generating graphics display subroutines from the terminal keyboard. Los Angeles: Behavioral Technology Laboratories (in press).

Rumelhart, D. W., Lindsay, P. H., & Norman, D. A. A process model for long-term memory. In E. Tulving and W. Donaldson (Eds.), Organization of Memory. New York: Academic Press, 1972.

Sheehan, P. W. Functional similarity of imaging to perceiving: Individual differences in vividness of imager. Perceptual and Motor Skills, 1966, 23, 1011-1033.

Sheehan, P. W. The role of imagery in incidental learning. British Journal of Psychology, 1971, 62, 235-243.

Wollmer, R. D. A Markov decision model for computer-aided instruction. Los Angeles: Behavioral Technology Laboratories, December 1973.

APPENDIX A

RIO INITIAL DATA CAPTURE DEFINITION

	No	Digits	
1	@	12	PP
2	A	12	PH
3	B	12	PS
4	C	12	PE
5	E	12	SE
6	F	12	SS
7	G	12	SH
8	H	12	SP
9	D	9	Turn 360°
10	T	12	Turn to a heading
11	J	9	Ease
12	N	9	Tighten
13	L	9	Level
14	Y	6	Correct answer
15	X	6	Used Answer Key
16	V	9	Tote Board Completion
17	M	0	Change Modes Static to Dynamic
18	P	6	Pause
19	V	9	Triangle
20	W	9	Tote Board
21	S	21	Fire
22	⊏	9	Left Sector Scan
23	⊐	9	Right Sector Scan
24	↑	9	Center Sector Scan
25	\	9	Brake Sector Scan
26	Q	12	Start Problem
27	K	12	Start Free Fly Problem
28	Z	0	Stop Problem
29	>	9	60→20 mile B-Scan Scale
30	<	9	20←60 mile B-Scan Scale
31	=	9	Lock One
32	;	9	Break Lock
33	?	9	Attempt Lock On - Fail

ILLUSTRATIONS OF STUDENT DATA STRINGS

1-8 @, A, B, C, E, F, G, H
Define turn severity and direction. If one of these 8 Alphas is followed directly by 12 digits (not by a D or T) the turn command was to turn a specified number of degrees.

e.g. $A\ \ \underset{1}{\overparen{230}}\ \ \underset{2}{\overparen{006}}\ \ \underset{3}{\overparen{040}}\ \ \underset{4}{\overparen{064}}$

1. Initial Fighter heading	230°
2. Range	6 miles
3. Turn specification	40°
4. Problem time	64 seconds

A specifies a port hard turn.

9 D
Define turn 360° command, preceded by one of Alphas 1-8 (@, A, B, C, E, F, G, H) for direction and severity specification.

e.g. $GD\ \ \underset{1}{\overparen{007}}\ \ \underset{21}{\overparen{160}}\ \ \underset{3}{\overparen{145}}$

GD turn starboard hard, 360°

1. Range	7 miles
2. Initial Fighter Heading	160°
3. Problem time	145 seconds

10 T
Define a turn to a specified heading. T is always preceded with an Alpha 1-8 (@, A, B, C, E, F, G, H) to specify turn severity and direction.

e.g. $FT\ \ \underset{1}{\overparen{170}}\ \ \underset{2}{\overparen{014}}\ \ \underset{3}{\overparen{210}}\ \ \underset{4}{\overparen{075}}$

FT starboard standard turn to specified heading

1. Initial Fighter heading	170°
2. Range	14 miles
3. Turn to specified heading	210°
4. Problem time	75 seconds

11-13 J, N, L
Ease, tighten, level

Turn is in progress when these functions used; the turn is modified as per indication.

e.g. N $\underset{1}{\overset{\frown}{002}}$ $\underset{2}{\overset{\frown}{350}}$ $\underset{3}{\overset{\frown}{090}}$

N Tighten turn

1. Range	2 miles
2. Current Fighter heading	350°
3. Problem time	90 seconds

14-15 Y, X

Y correct answer entered
X answer key used

Number of errors and latency stopped

e.g. Y $\underset{1}{\overset{\frown}{002}}$ $\underset{2}{\overset{\frown}{012}}$

Y correct answer entered

1. No errors	2
2. Latency	12 seconds

16 U Tote board completion

e.g. U $\underset{1}{\overset{\frown}{020}}$ $\underset{2}{\overset{\frown}{003}}$ $\underset{3}{\overset{\frown}{046}}$

U Tote board completion

1. Range	20 miles
2. No errors	3
3. Time to completion of tote board	46 seconds

17 M indicates transfer fro static to dynamic phase of dynamic problem, problem discarded from analysis.

e.g. MU $\overset{\frown}{020}$ $\overset{\frown}{003}$ $\overset{\frown}{046}$

Interpreted as U above but problem is discarded from analysis

18 P Pause
Use of pause key

e.g. $P\ \underset{1}{\overset{\frown}{090}}\ \underset{2}{\overset{\frown}{010}}$

1. Time pause terminated	90 seconds
2. Duration of pause	10 seconds

19 V Triangle
Geometric display of intercept triangle

e.g. $V\ \underset{1}{\overset{\frown}{006}}\ \underset{2}{\overset{\frown}{005}}\ \underset{3}{\overset{\frown}{080}}$

1. Range	6 miles
2. Duration of triangle use	5 seconds
3. Problem time at end of use of triangle	80 seconds

20 W Tote Board
Used only in dynamic phase of dynamic problem.
Displays updated version of tote board variables.

e.g. $W\ \underset{1}{\overset{\frown}{007}}\ \underset{2}{\overset{\frown}{012}}\ \underset{3}{\overset{\frown}{065}}$

1. Range	7 miles
2. Duration of use	12 seconds
3. Problem time at end of use	65 seconds

21 S Fire
Indicates the termination of a dynamic or free fly problem.

e.g. $S\ \underset{1}{\overset{\frown}{001}}\ \underset{2}{\overset{\frown}{100}}\ \underset{3}{\overset{\frown}{145}}\ \underset{4}{\overset{\frown}{001}}\ \underset{5}{\overset{\frown}{174}}\ \underset{6}{\overset{\frown}{001}}\ \underset{7}{\overset{\frown}{002}}$

1. Range	1 mile
2. Hit probability	100%
3. Problem time	145 seconds
4. Range	1 mile
5. Target aspect	174°
6. Angle off	1°
7. Number of turns	2

22-25 ⊏ , ⊐ , ↑
Left, right, center sector scan
and \ leave sector scan

e.g. ⊏ $\begin{array}{ccc}\overset{\frown}{009} & \overset{\frown}{009} & \overset{\frown}{048} \\ 1 & 2 & 3\end{array}$

⊏ Enter left sector scan

1.	Range	9 miles
2.	Range	9 miles
3.	Problem time	48 seconds

26-27 Q, K
Q Start of static or dynamic problem
K Start of free fly problem

e.g. Q $\begin{array}{ccccc}\overset{\frown}{029} & \overset{\frown}{144} & \overset{\frown}{0} & \overset{\frown}{1100} & \overset{\frown}{6} \\ 1 & 2 & 3 & 4 & 5\end{array}$

Q Problem start

1.	Student number	29
2.	Problem number	144
3.	0 $\Rightarrow$ dynamic problem A 1 here and 0 in 5 would be static	
4.	Criterion word 1100 $\Rightarrow$ 2 of 4 TA categories have been consecutively successfully completed	
5.	6 $\Rightarrow$ dynamic Speed level	460 kts

28 Z indicates end of session

29-30 < , > B-Scan range change

< 20 $\rightarrow$ 60 mile range
> 60 $\rightarrow$ 20 mile range

e.g. $\begin{array}{ccc}\overset{\frown}{017} & \overset{\frown}{017} & \overset{\frown}{020} \\ 1 & 2 & 3\end{array}$

> change from 60 to 20 mile range scale

1.	Range	17 miles
2.	Range	17 miles
3.	Problem time	20 seconds

31-33 = , ; , ?
Lock on, break lock, attempt and fail to lock on.

e.g. $= \underset{1}{\overset{\frown}{004}}\ \underset{2}{\overset{\frown}{015}}\ \underset{3}{\overset{\frown}{070}}$

= Lock on radar

1. Range	2 miles
2. Angle Off	15°
3. Problem time	70 seconds